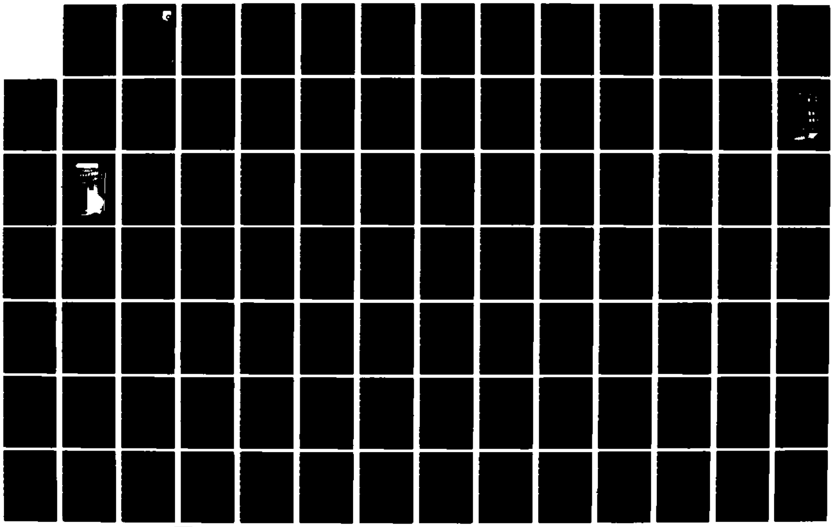
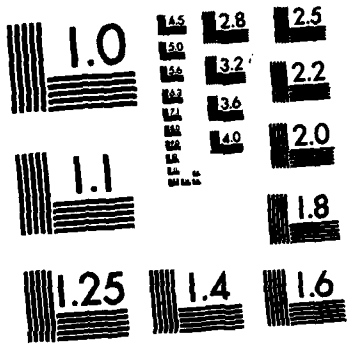
AD-A124 621 THE REMOTE LINK UNIT: A DEMONSTRATION OF OPERATIONAL PERFORMANCE PART III..(U) HOUSTON UNIV TX DEPT OF ELECTRICAL ENGINEERING C J TAVORA ET AL. AUG 81 1/3

UNCLASSIFIED AFWAL-TR-81-1131-PT-3-1 F33615-80-C-1095 F/G 17/2 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

②

AFWAL-TR-81-1131
PART III, VOLUME 1

THE REMOTE LINK UNIT: A DEMONSTRATION OF OPERATIONAL PERFORMANCE

Part III - Design Manual
Volume 1

C. J. Tavora
J. R. Glover, Jr.
M. A. Smither

Electrical Engineering Department
University of Houston
4800 Calhoun Boulevard
Houston, Texas 77004

August 1981

Final Report for Period 1 April 1980 to 31 December 1980

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

[illegible]
PHILIP C. GOLDMAN
Project Engineer
Information Transfer Group
Avionics Laboratory

Donald L. Moon
DONALD L. MOON, Chief
Information Processing Technology Branch
Avionics Laboratory

FOR THE COMMANDER

Richard H Boivin
RICHARD H. BOIVIN, Colonel, USAF
Chief, System Avionics Division
Avionics Laboratory

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-81-1131, Part III, Vol 1	2. GOVT ACCESSION NO. AD-A124 621	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE REMOTE LINK UNIT: A DEMONSTRATION OF OPERATIONAL PERFORMANCE PART III - Design Manual Volume 1		5. TYPE OF REPORT & PERIOD COVERED Final Report for Period 1 Apr 80 to 31 Dec 80
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C.J. Tavora, J.R. Glover, Jr., M.A. Smither, M.H. Collins, W.C. Law, P.D. Balsaver, H.C. Hsia, and T.T. Lin		8. CONTRACT OR GRANT NUMBER(s) F33615-80-C-1095
9. PERFORMING ORGANIZATION NAME AND ADDRESS Electrical Engineering Department University of Houston 4800 Calhoun, Houston, TX 77004		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62204F 2003 08 07
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory (AFWAL/AAAT-3) AF Wright Aeronautical Laboratories (AFSC) Wright-Patterson AFB, OH 45433		12. REPORT DATE August 1981
		13. NUMBER OF PAGES 229
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Remote Link Unit, Remote Terminals, Universal Interface, Electronic Nameplate, Distributed Avionics, Fault Monitoring, Fault Recording.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is Part III, Design Manual for the RLU Demonstration System, and provides detailed information on the hardware and software design. It is in two volumes: Volume 1 contains the description of the design, and Volume 2 contains the Appendices. Accompanying reports are Part I Summary and Part II User's Manual.

PREFACE

This document was prepared by the University of Houston, Houston, Texas, on Air Force Contract No. F33615-80-C-1095, entitled "The Remote Link Unit - A Demonstration of Operational Performance."

The work was administered under the direction of the Information Transfer Group, Information Processing Technology Branch, System Avionics Division of the Avionics Laboratory, under Project 2003, "Avionic System Design Technology," Task 08, "Multiplex and Information Transfer Technology," Work Unit 07, "Remote Link Unit Demonstration." The work was performed during the period 1 April 1980 to 31 December 1980 and this report was submitted in August 1981. The Air Force Project Engineer was Philip C. Goldman (AFWAL/AAAT-3).

The work is a continuation of a previous feasibility study entitled, "The Remote Link Unit: An Advanced Remote Terminal for MIL-STD-1553A." The results of this study are documented in a technical report entitled, "Remote Link Unit Functional Design: An Advanced Remote Terminal for MIL-STD-1553B," which was published as AFAL-TR-79-1176, AD-A080126. An add-on to this previous study resulted in a second technical report entitled, "The Remote Link *Unit: Applications to* the Design for Repair Methodology Program," published as AFWAL-TR-80-1033, AD-A086126.

This report summarizes the design, development, and testing accomplished under the contracted work. The Principal Investigator and Program Manager was Dr. Carlos J. Tavora. Drs. John Glover, Jr. and Miles A. Smither were Co-Investigators. Dr. Tavora was responsible for the system architecture and modularization of the design. Dr. Glover supervised the design of the software for the Link Manager Simulator and the Link Module. He was assisted by Messrs. Hao-Cheng Hsia, William C. Law, and Parmanand Balsaver. Dr. Smither was assisted by Mr. Tzer-Tsan Lin in the design of the Interface Configuration Adapter. Mr. H. Mitchell Collins was in charge of the design of the Electronic Nameplate and the Nameplate Interface Controller.

This report is organized in three parts: Part I - Summary, Part II - User's Manual, and Part III - Design Manual. Part III is separated into two volumes: Volume 1 is the main body of the Design Manual, while Volume 2 contains the appendices.

This is Volume 1 of Part III, Design Manual. It describes the detailed hardware and software design of the RLUDS, and is organized such that major sections relate to each functional subsystem within the RLUDS.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONT'D.)

LIST OF ILLUSTRATIONS (CONT'D.)

LIST OF ILLUSTRATIONS (CONT'D.)

LIST OF TABLES

LIST OF TABLES (CONT'D.)

LIST OF ACRONYMS

AC	Alternating Current
A/D	Analog to Digital
BCD	Binary Coded Decimal
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRT	Cathode Ray Tube
D/A	Digital to Analog
DC	Direct Current
DIP	Dual In-line Package
DMA	Direct Memory Access
EAROM	Electrically Alterable ROM
EEPROM	Electrically Erasable Programmable ROM
EPROM	Erasable Programmable ROM
FP	Front Panel
ICA	Interface Configuration Adapter
I/O	Input/Output
ISR	Interrupt Service Routine
LA	Link Address
LM	Link Module
LMG	Link Manager
LMP	LM Processor
LSB	Least Significant Bit
MP	Maintenance Port
MSB	Most Significant Bit
MUX	Multiplexer
NIC	Nameplate Interface Controller
NP	Nameplate
PCB	Printed Circuit Board
PIA	Peripheral Interface Adapter
PROM	Programmable Read Only Memory
RAM	Random Access Memory
RLU	Remote Link Unit
RLUDS	Remote Link Unit Demonstration System
RMW	Read-Modify-Write
ROM	Read Only Memory
RT	Remote Terminal
SIC	Subsystem Information Channel
SM	Shared Memory
SRU	Shop Replaceable Unit
TTL	Transistor-Transistor Logic
UFT	User File Table

SECTION 1

INTRODUCTION

This document is a design manual for the Remote Link Unit Demonstration System (RLUDS). The Remote Link Unit (RLU) is a new design concept for remote terminals. This document contains detailed design information on the RLUDS. Design and implementation of the RLUDS was performed for the Air Force Wright Aeronautical Laboratories under contract #F33615-80-C-1095.

1.1 THE RLU DEMONSTRATION UNIT

The RLUDS described in this design manual is an operational hardware breadboard prototype that performs most of the important RLU functions. The RLUDS is not intended to be a complete RLU ~~prototype~~ but it demonstrates the most unique parts of the RLU. This effort has demonstrated the feasibility of implementation of the Link Module (LM), the Interface Configuration Adapter (ICA), the Electronic Nameplate (NP) and the Interface between the Link Module and the Link Manager. The Link Manager (LMG) was simulated with a PDP-11 computer. The extent of RLU implementation encompassed by the demonstration unit is illustrated in Figure 1. The detailed design of the RLUDS is based on the functional design described in the document, "Remote Link Unit Functional Design: An Advanced Remote Terminal for MIL-STD-1553B" technical report AFAL-TR-79-1176. The configuration of the RLUDS is presented in Figure 2.

1.2 DESIGN CONSTRAINTS

In order to implement the RLUDS within a short time span, it was im-

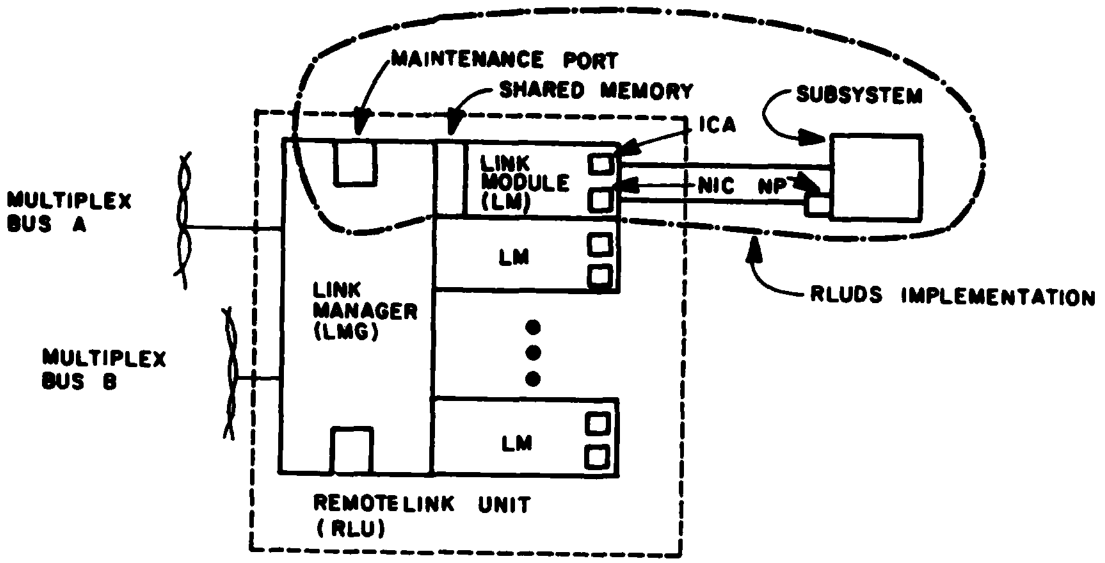


Figure 1 RLUDS Implementation of the RLU

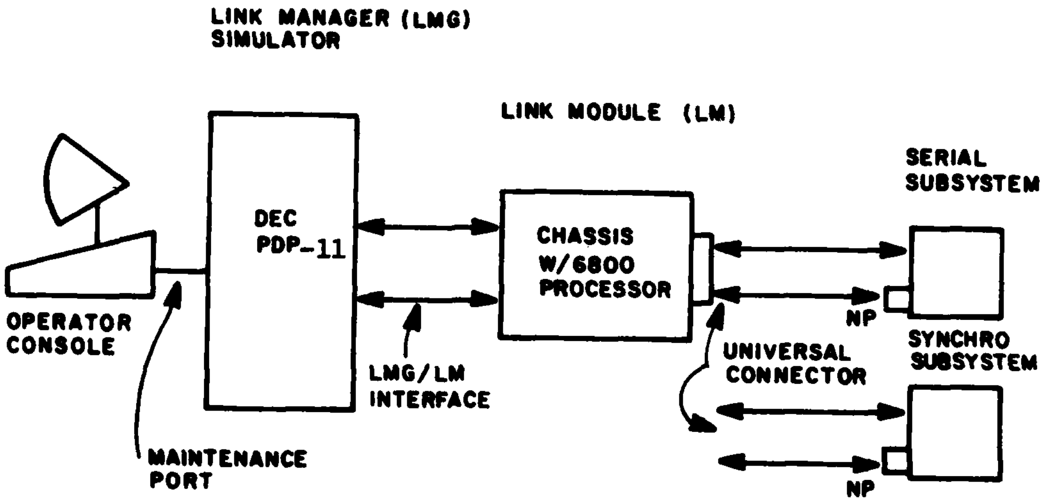


Figure 2 RLUDS Configuration

perative to limit the design effort to the development of components which were nonexistent and constituted critical elements for verification of the RLU concepts. In order to accommodate use of off-the-shelf components, it was necessary for the RLUDS design to deviate from the RLU functional design (document AFAL-TR-79-1176). Some of the required deviations are identified in the statement of work for the RLUDS. Other deviations have been identified in the process of designing the demonstration system. In each case, a deviation to the functional design was allowed when it related a change that did not alter the RLU concept to be evaluated, but rather represented a design scaling of electrical or timing dimensions. The most significant difference between the RLU functional design and the RLUDS stems from the use of an 8 bit microprocessor for implementation of the Link Module. This selection,which was dictated by the available microprocessor development facilities at the University of Houston,led to an 8-bit LM internal bus instead of the 16-bit bus described in the functional design. Use of the 8-bit LM internal bus has caused corresponding dimensional changes in the Interface Configuration Adapter, the Nameplate Interface Controller and the Shared Memory.

1.3 TEST PROCEDURE

A test plan that outlines the approach used to demonstrate the operational performance of the RLU is presented in the document entitled, "A Test Plan for the Remote Link Unit Demonstration System." This test plan describes a three part test procedure that demonstrates the operation of the Interface Configuration Adapter, the Subsystem Information Channel and the Remote Link Unit.

1.4 *ORGANIZATION OF THE MANUAL*

This manual has been organized in a manner that simplifies the documentation of the RLUDS design. Sections 2 through 5 describe the major RLUDS components in terms of the theory of operation, the hardware design, the software design and the test procedure. Section 2 describes the Link Module. Section 3 describes the design of the Interface Configuration Adapter. Section 4 describes the Subsystem Information Channel which includes the Nameplate Interface Controller, the nameplate bus and the Electronic Nameplate. Section 5 describes the Link Manager simulator and the software required to support the RLUDS demonstration. Section 6 describes the serial and synchro subsystems used in the RLU demonstration. The detailed hardware diagrams and parts lists are contained in Appendix B . The detailed software description is presented in Appendix C .

SECTION 2

LINK MODULE

2.1 DESIGN PHILOSOPHY

2.1.1 DESCRIPTION

The Link Module (LM) consists of a parallel bus structure with Motorola 6800 Processor, PROM, RAM, and three separate interfaces as shown in Figure 3. Each interface connects to the common bus structure of the processor chassis. There is a Shared Memory (SM) interface to the Link Manager (LMG) through which all communication between the LM and the LMG takes place. There is a Nameplate Interface Controller (NIC) interface to the subsystem nameplates for serial communication between the LM and the NP's on the Subsystem Information Channel (SIC) bus. There is a parallel digital interface to the Interface Configuration Adaptor (ICA) for communication between the LM and the ICA.

2.1.2 DESIGN PHILOSOPHY

The Link Module (LM) is the intelligent link between the Link Manager (LMG) and the subsystems. It consists of a processor with three interfaces.

The following design goals were established for the Link Module design:

1) Minimize hardware fabrication.
2) Utilize off-the-shelf items.
3) Concentrate development effort on conceptual features of the RLU not yet demonstrated as feasible.
4) Provide built-in trouble shooting capability through the use of

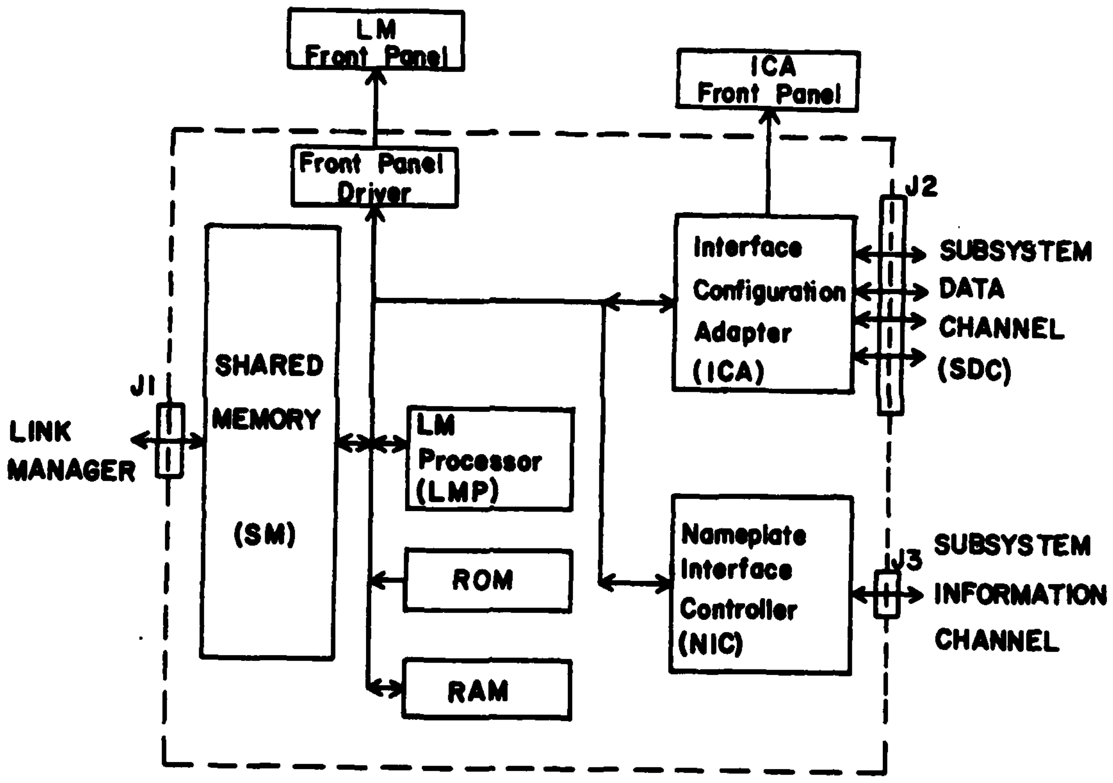

LM Front Panel
ICA Front Panel
Front Panel Driver
J2
Interface Configuration Adapter (ICA)
SUBSYSTEM DATA CHANNEL (SDC)
SHARED MEMORY (SM)
J1
LINK MANAGER
LM Processor (LMP)
ROM
RAM
Nameplate Interface Controller (NIC)
J3
SUBSYSTEM INFORMATION CHANNEL

Figure 3 Link Module Hardware Architecture

- Modularized enclosures,
- Front panel status indicators, and
- A software monitor debugger.

2.1.3 IMPLEMENTATION

The microprocessor 6800 was chosen for the LM implementation since it met the processing requirements and substantial hardware and software development support for this processor is available at the Digital Control and Automation System Laboratory at the University of Houston.

The LM is distributed among two chassis as shown in Figure 4 . Except for the ICA which is housed in the top chassis all other LM components are housed in the main chassis. The main chassis is a Motorola chassis with a card cage and bus system that accepts a variety of off-the-shelf modules for the 6800 system.

2.2 HARDWARE DESIGN

2.2.1 LINK MODULE HARDWARE DECOMPOSITION

The Link Module (LM) hardware comprises two chassis: main chassis and extension chassis. The main chassis holds power supplies, card cage with various cards (the processor, program memory, data memory, Shared Memory, Nameplate Interface Controller, bus connections to the Interface Configuration Adaptor), front panel and back panel.

The extension chassis houses the ICA and mounts directly on top of the main chassis and contains 3 circuit boards, a front panel and a back panel. The ICA front panel displays ICA configuration information which is useful for monitoring the interface operation. The back panel is used for power and signal connections.

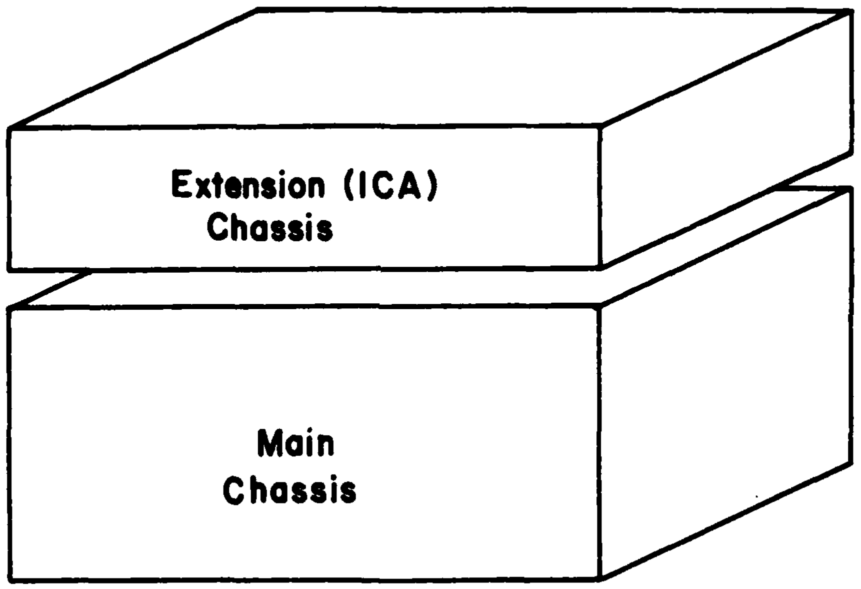


Figure 4 Link Module Enclosures

The cards in the main chassis are listed in Table 1 . The chassis (Motorola P/N M68MMLC) contains a triple DC output power supply (+5V, +12V, -12V) and a 10 slot card cage. Two additional voltage supplies have been installed in the main chassis: a dual DC output (+15V, -15V) for the ICA, and a single DC output (+25V) for the NP. The CPU card (Motorola P/N M68MM01A2) contains a Motorola 6800 eight bit processor, 1 MHz crystal controlled clock, 1K byte read/write memory (not used in the LM implementation), four sockets for 2716 EPROMs (2K bytes each of program memory), four eight bit parallel digital ports (not used in the LM implementation), and one serial RS-232C data terminal interface (used by the M68MM08A ROM for system debugging). A program memory card (Motorola P/N M68MM04A) provides sixteen sockets for 2716 EPROMs for 32K bytes of additional program memory space. Data memory is provided by a Motorola MEX6816-1HR which has 16K bytes of read/write memory. Three custom interface cards (SM, NIC, FP/ICA) are based on Motorola MEX68USM universal interface cards which provide address decoding and bus buffering.

The extension chassis houses the ICA. This chassis has three circuit cards: a control/processor interface module and two signal I/O interface cards (one for each ICA group). The ICA extension chassis fits physically on top of the LM main chassis.

A front panel layout is shown in Figure 5 . Located on the front panel are various display indicators, facility to write into Shared Memory, power on/off switch, and a reset button. Additionally, the extension chassis front panel has various ICA status indicators.

The main chassis back panel has the AC power input, three fuses (115VAC to LM, +5VDC to NP, +25VDC to NP), a fan for air circulation, a

Table 1

LINK MODULE MAIN CHASSIS CARDS

Slot #	Card #	Description	Connectors
1	1	FP & ICA	INTCBL 5ØØ to Front Panel & INTCBL 2ØØ to ICA.
2	-	Empty	------
3	3	PROM	None
4	4	RAM	None
5	-	Empty	------
6	6	NIC	INTCBL 3ØØ to NP.
7	-	Empty	------
8	8	CPU	INTCBL 4ØØ to CRT.
9	9	SM	INTCBL 1ØØ to LMG.
1Ø	-	Empty	------

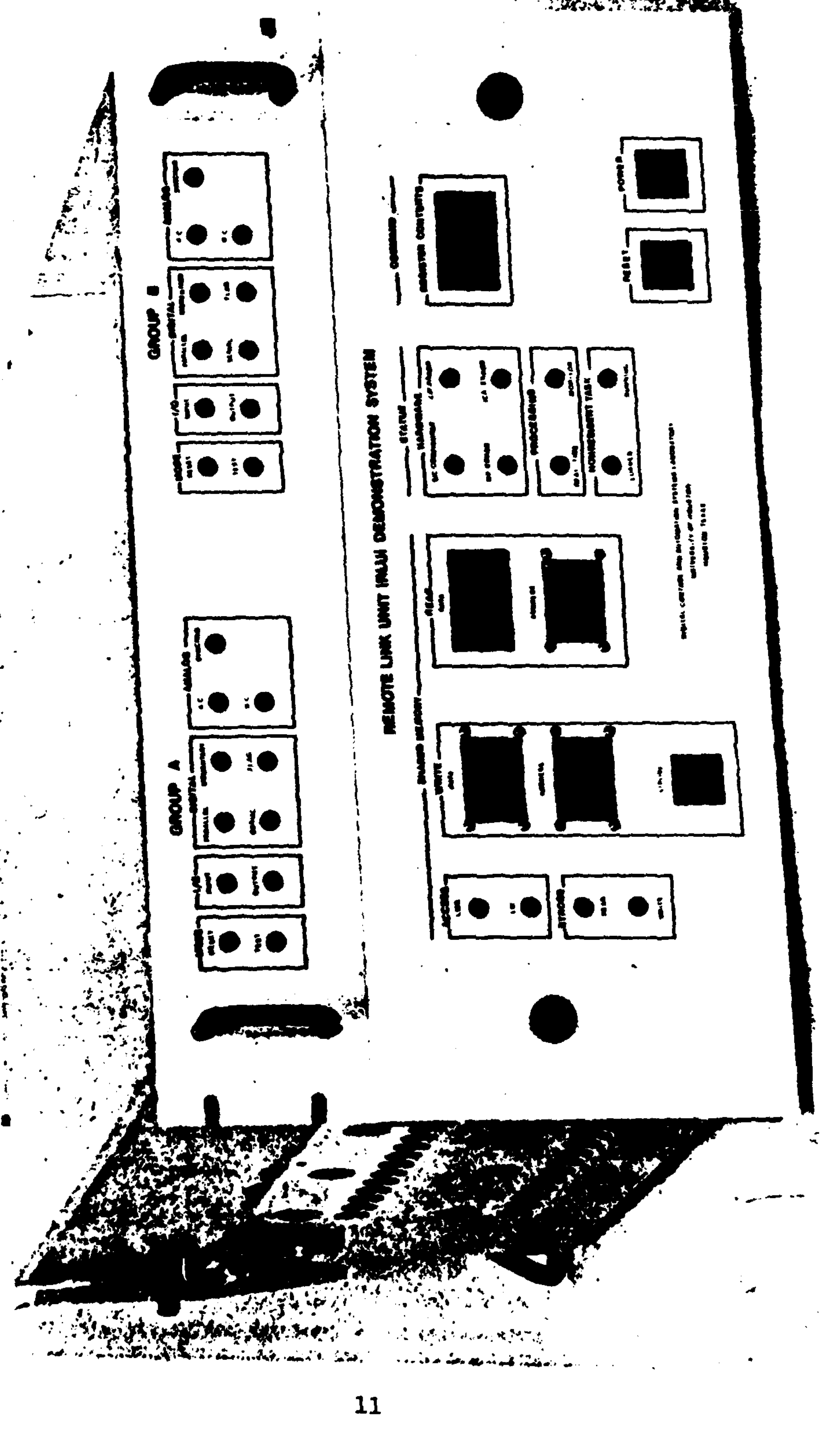


Figure 5 LM Front Panel

connector to the LMG for communication through Shared Memory, a connector to the Electronic Nameplate, connector to a data terminal for system debugging, and a toggle switch for choosing the restart vectors (auto or monitor). The extension chassis back panel has three fuses (+5VDC, +15VDC, -15VDC) and a connector to the subsystem.

2.2.2 MODULES AND ADDRESS ASSIGNMENTS

The address assignment for each LM module is shown on the memory map presented in Figure 7 which shows the Motorola M6800 address space of 64K bytes. All addresses shown in this figure are expressed as hexadecimal numbers (base 16). The 16K data memory is in the lower addresses from ØØØØ to 3FFF, with the 1K byte on the CPU card disabled to avoid address conflicts. The program memory card has two independent 16K blocks. The first block (addresses 4ØØØ to 7FFF) contains the software tasks EXEC, UPDATE, ISR, CMDITR, part of NPHND, and ICAHND. The second block uses only one quarter of its address space (88ØØ to 97FF) and contains the software tasks SMHND and the rest of the NPHND. The SM and NP fit into addresses 8ØØØ to 83FF. Addresses 84ØØ to 87FF are used for parallel and serial I/O on the CPU card. The ICA and front panel interface modules fit into addresses 98ØØ to 98FF. The Motorola M68MM08A MICRObug ROM monitor resides in addresses F8ØØ to FFFF (a back panel switch may be used to enable the monitor mode of operation).

2.2.3 SHARED MEMORY INTERFACE CARD

The Link Manager (LMG) and the Link Module (LM) perform all of their communications through a shared memory interface consisting of 256 bytes of read/write memory. The LM makes its accesses to SM directly,

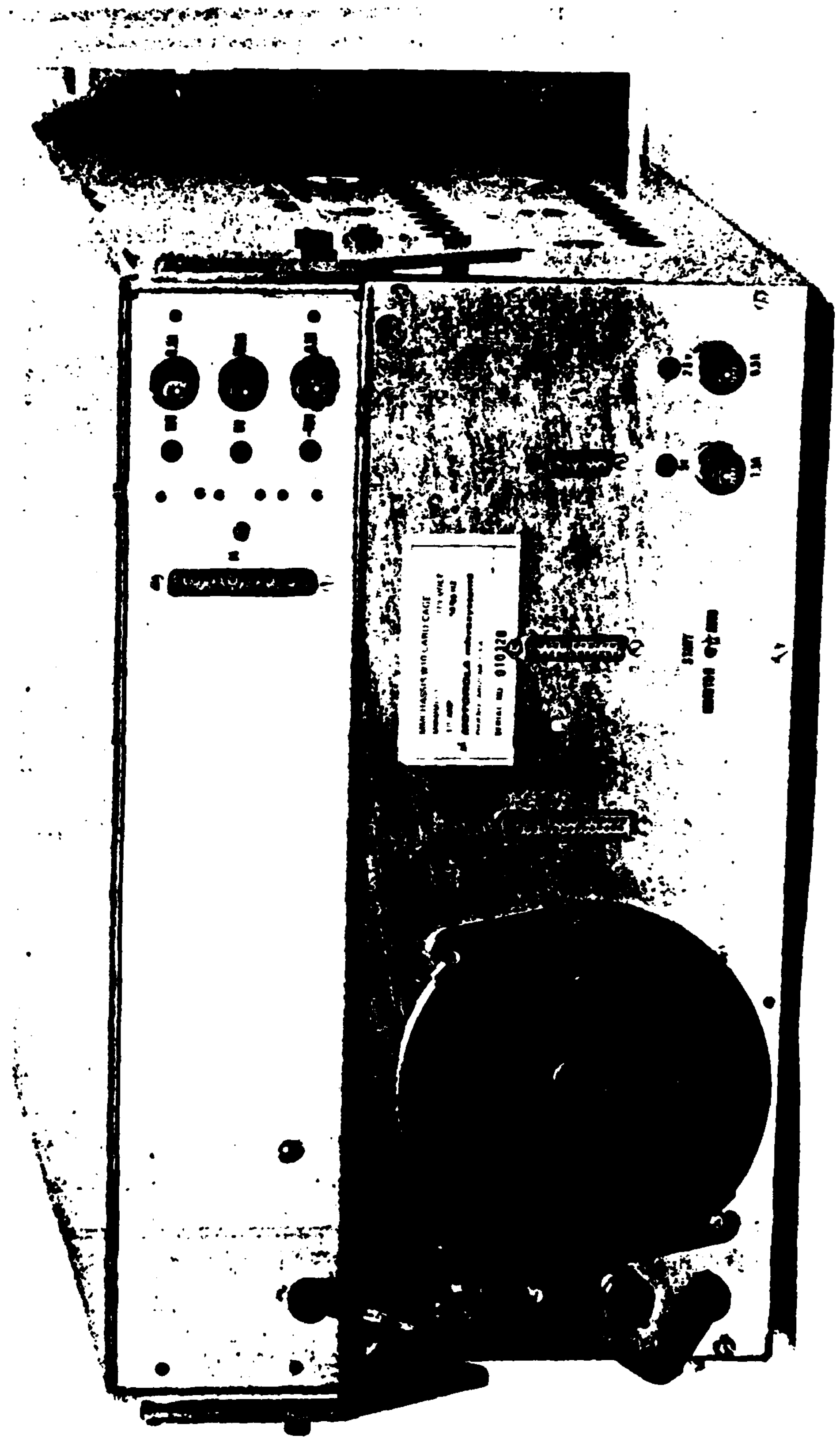


Figure 6 LM Back Panel

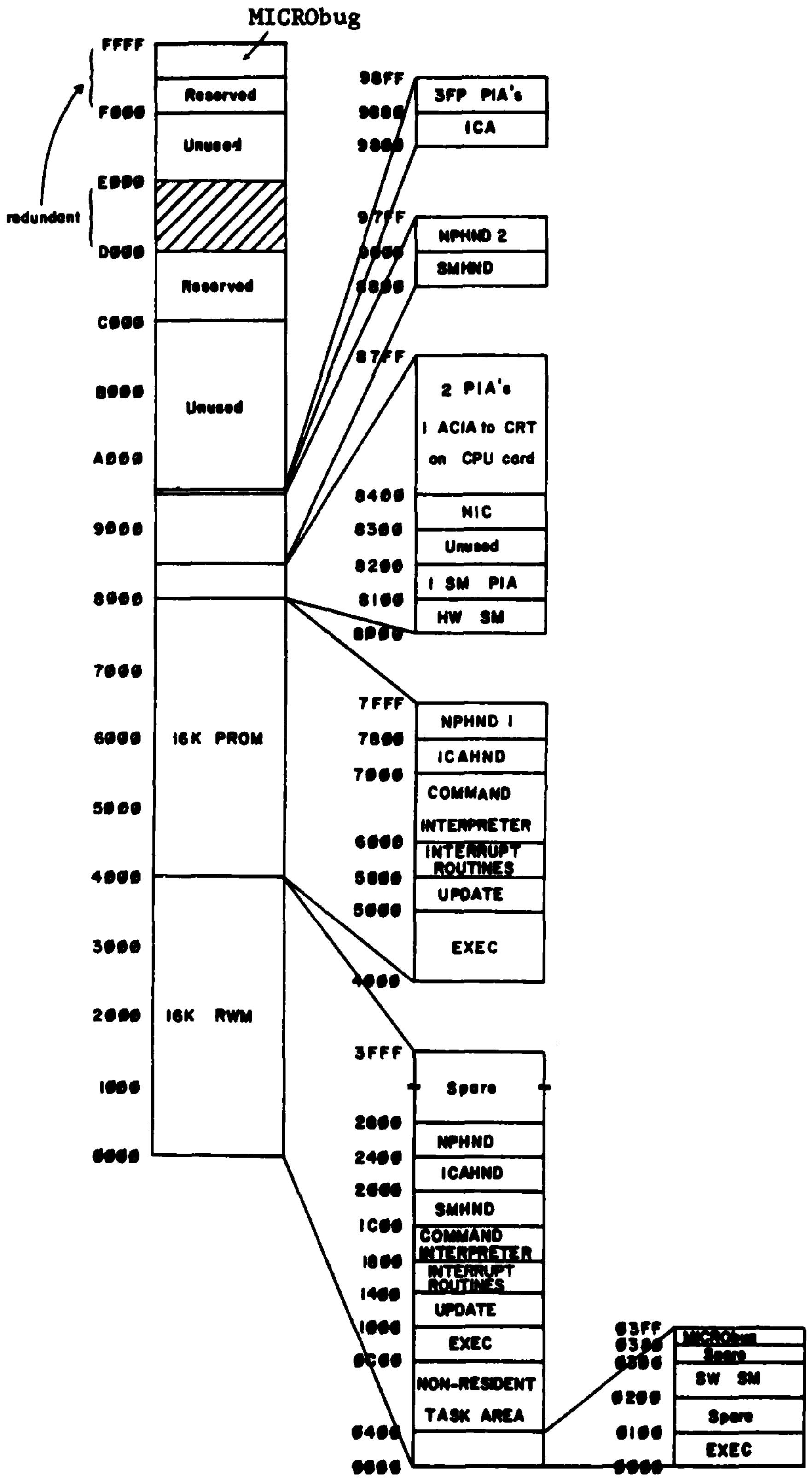


Figure 7 LM Memory Map

just as it does to any other memory location, since the SM is in the memory space of the LM (at addresses 8ØØØ to 8ØFF). The LMG which is simulated on a DEC PDP 11/70 interfaces to the LM Shared Memory (SM) through a DEC DR11C board as block diagrammed in Figure 8 . All transfers are single 8-bit byte transfers and are done with a complete handshake for each byte using three of the four control lines shown in Figure 8 .

The LMG interface board has two output control lines labeled CSRØ and CSR1. These are used as Chip Select (CS) and Write Enable (WE) respectively to the SM hardware. The LMG interface board has two input control lines labeled AREQ and BREQ. The BREQ line is unused, while the AREQ line is used for the return handshake for all data transfers. The corresponding handshakes for read (RD), write (WR), and read-modify-write (RMW) are diagrammed in Figure 9 . A complete handshake is performed for each byte of data transferred between the LM and LMG.

The SM address and data buses are shared between the LM and LMG and thus cannot be accessed by both at the same time. This is resolved by making the LMG accesses to SM by Direct Memory Access (DMA). This DMA is implemented by having the LMG CS line act as a halt request to the LM 6800 processor. The Bus Available (BA) signal from the 6800 is then used as the returning handshake AREQ to the LMG. Details of this timing can be seen in Figure 10 .

RMW is a special implementation which allows each side to read a data byte, modify the value, and write the value back without the other side being able to access it during this time. For LMG accesses this is no problem using the special RMW handshake shown in Figure 9 since the LM is halted while LMG is accessing. For the LM a special procedure is needed.

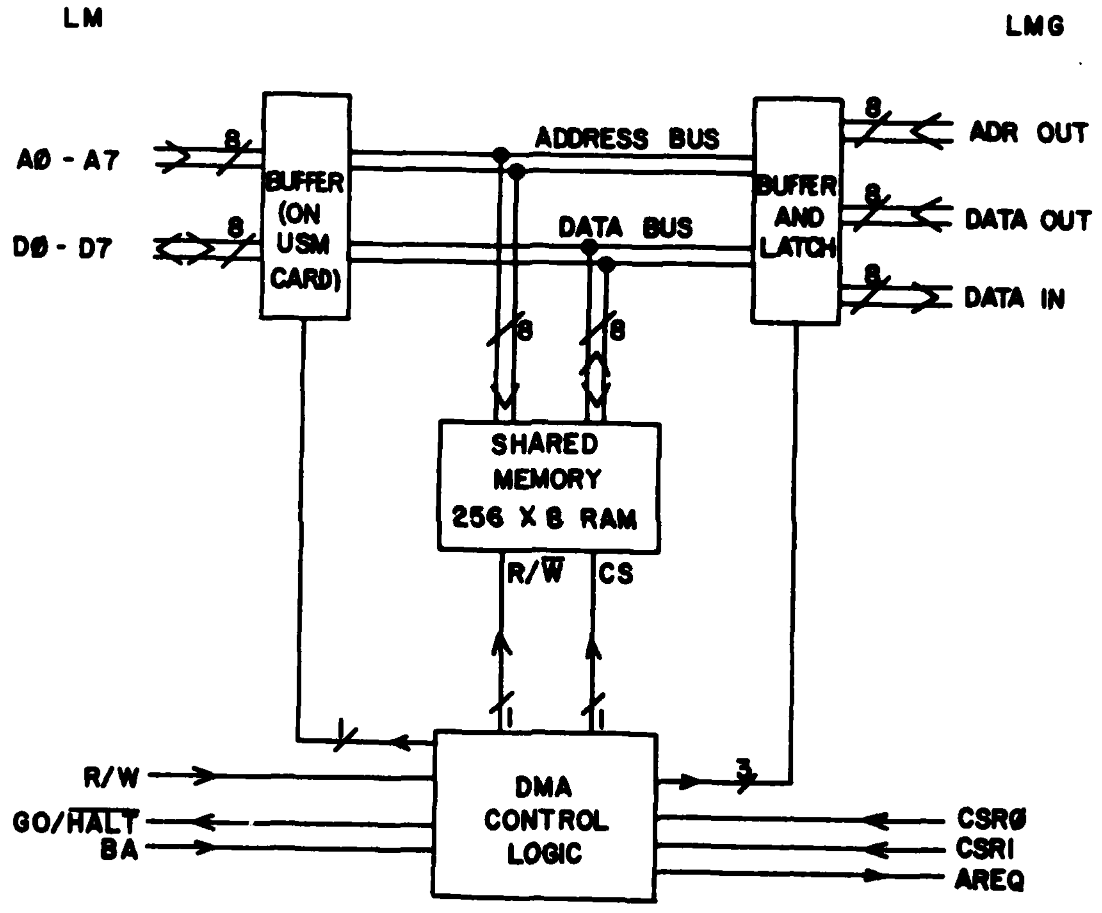
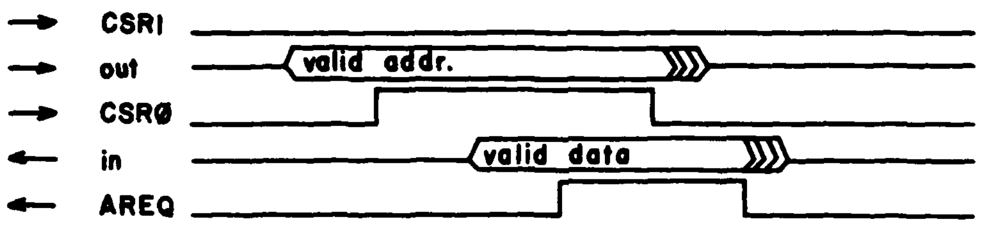
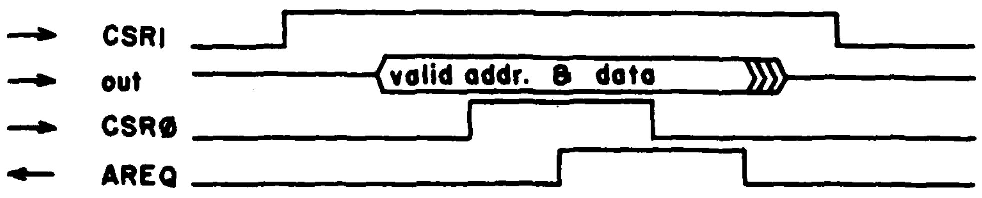


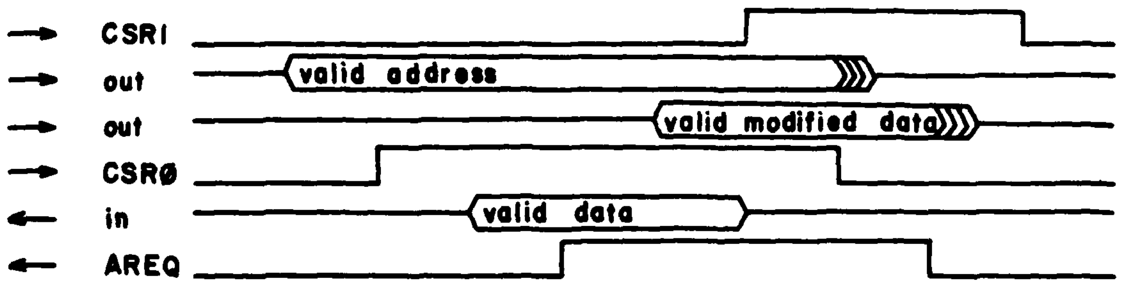
Figure 8 SM Block Diagram



(a)

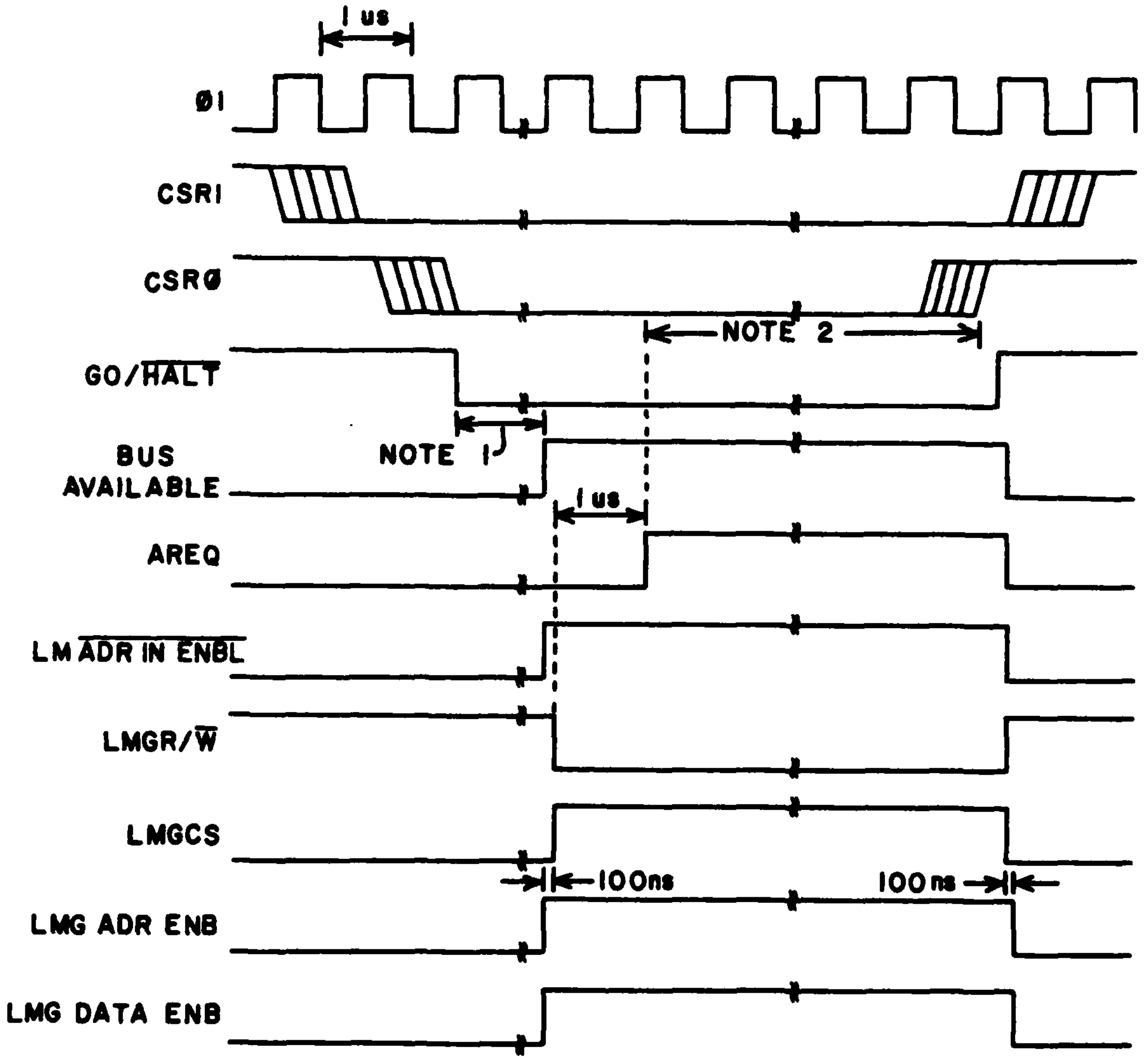


(b)



(c)

Figure 9 Handshake between the LMG and LM for (a) read, (b)write and (c)read-modify-write.


1 us
Ø1
CSR1
CSRØ
NOTE 2
GO/$\overline{\text{HALT}}$
NOTE 1
BUS AVAILABLE
1 us
AREQ
LM $\overline{\text{ADR IN ENBL}}$
LMGR/$\overline{\text{W}}$
LMGCS
100ns
100 ns
LMG ADR ENB
LMG DATA ENB

NOTES: 1 $0.1us < t < 12us$ DEPENDING ON STATUS OF CURRENT 6800 INSTRUCTION.

2 $0 < t < \infty$ SHOULD BE KEPT SHORT SINCE LM IS HALTED.

Figure 10 SM Timing Diagram

Built onto the SM card is a Peripheral Interface Adaptor (PIA) chip with an output pin (LMRMW) used to temporarily disable the LMG from accessing SM. When this LMRMW is high the LMG CS line cannot cause a halt request to the LM 6800 processor. Any LM software needing to do a RMW operation must first set LMRMW high, do the RMW operation, and set LMRMW low. If the LMG is attempting to access SM at this moment, it will simply appear as a slow response to the handshake.

Whenever the LMG issues a function command (by writing into SM location FF) or issues a data transfer command (by writing into SM location FE) an interrupt to the LM is generated. This is implemented by decoding the SM address bus with LMG CS and LMG WE and inputting these to interrrupt control pins on the PIA.

Four access strobes are generated and sent to the Front Panel (FP) card for display on the FP.

Detailed schematics can be found in Appendix B, Section 2-B.

2.2.4 FRONT PANEL/ICA CARD

The LM Front Panel (FP) pictured in Figure 5 is connected to a FP driver card as block diagrammed in Figure 11. This FP card consists of three Motorola Peripheral Interface Adaptors (PIA's) which each have two 8 bit parallel I/O ports. This is a total of six ports which are memory mapped in the LM. Thus the FP displays and switches are software driven by a subroutine in the LM which is called by the Update task. The FP information is only valid when the LM is running in a real-time mode of operation.

The bus connections to the ICA are on the same card as the FP PIA's. These are independent of the FP and are on the same card only for convenience. Detailed schematics can be found in Appendix B, Section 2-C.

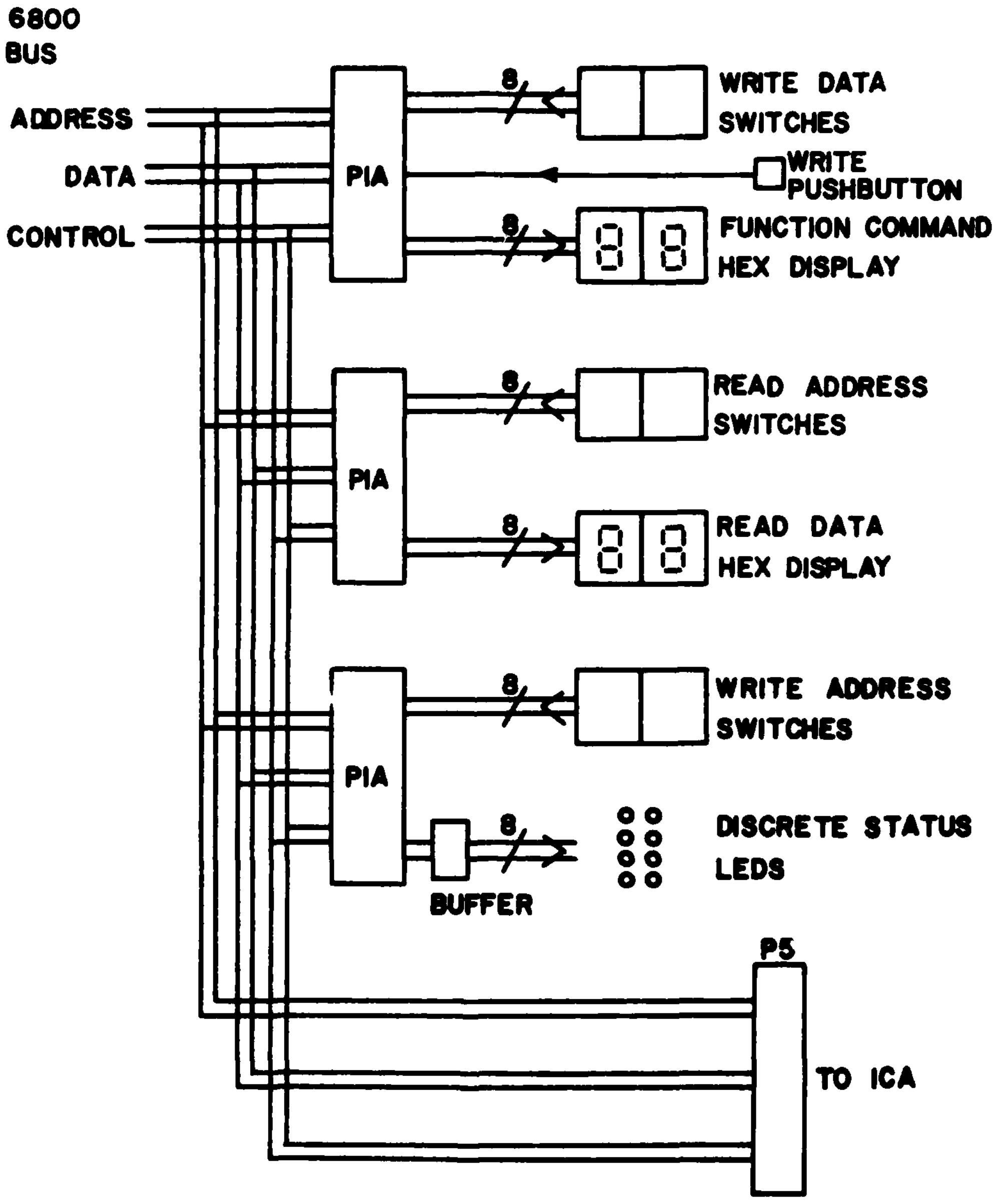


Figure 11 FP/ICA Card Block Diagram

2.2.5 NIC/TIMER CARD

The Nameplate Interface Controller (NIC) card, residing in the LM, enables programs running on the LM to communicate with electronic nameplates via the subsystem information channel bus. This card translates the LM's bus signals into the signals compatible with the SIC bus. Also installed in the NIC card is the LM's timer circuit which generates an interrupt to the LM processor every 10 milliseconds. The design of this card is described in detail in Section 4.3.1 of this document.

2.3 SOFTWARE DESIGN

Software in the Link Module (LM) consists of a real time executive and several tasks which implement the LM functions. Assembly language is used since the FORTRAN available for the 6800 system does not support the multi-tasking capability required. A top-down, modular approach to the software design has been used throughout. The LM software modules are described in the sections that follow.

2.3.1 LINK MODULE SOFTWARE DECOMPOSITION

The Link Module (LM) software is a multitasking system, with task scheduling controlled by a round robin executive. Each task is a self contained unit and performs a well defined function. However, one task may require the services of another task in order to complete its function. Intertask control interaction is achieved through calls to the executive. Intertask data sharing is implemented through global common variables. The tasks include an update timer, three handlers, a command interpreter, and the non-resident task. These are diagrammed in Figure 12 and listed in Table 2 .

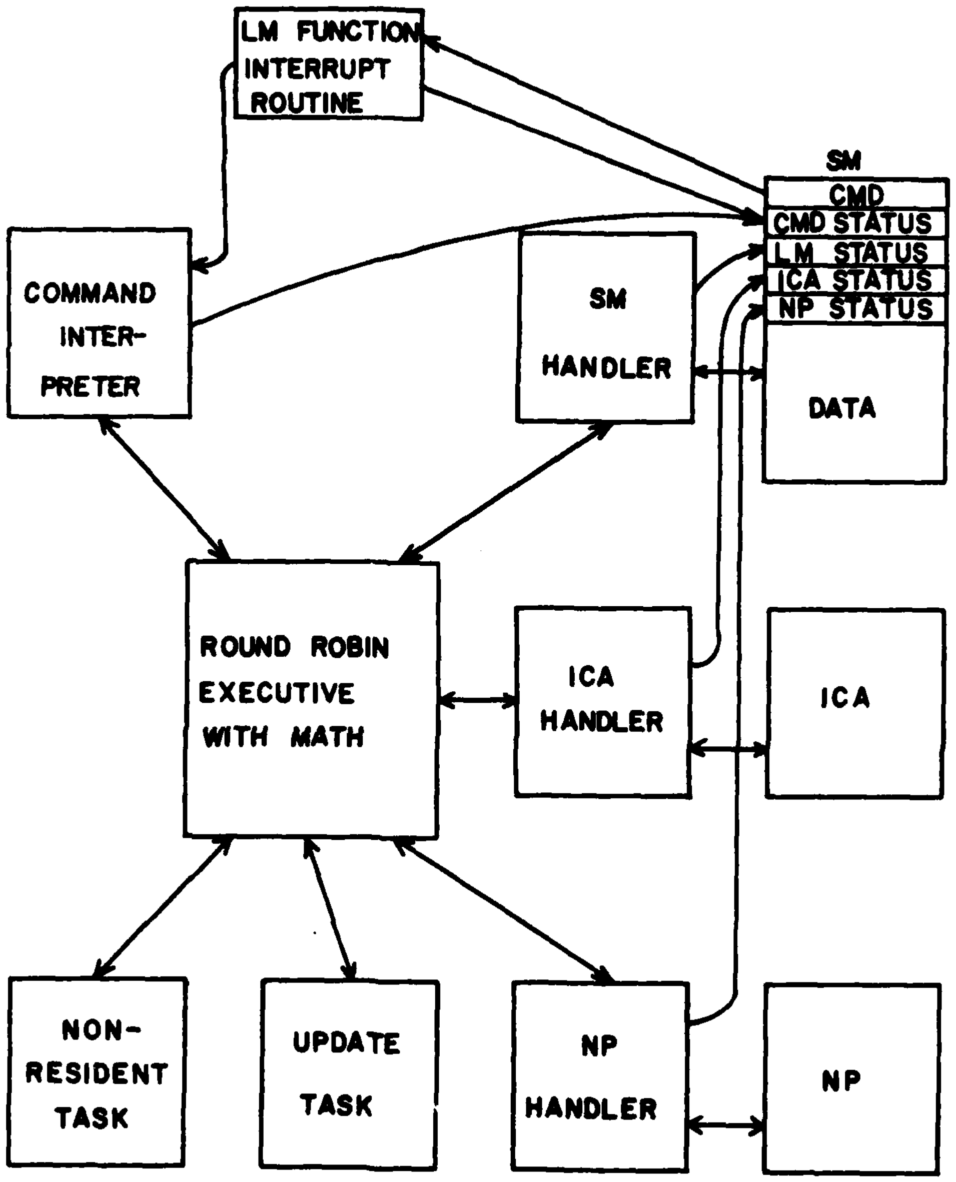


Figure 12 Executive Interactions

Table 2

LINK MODULE TASKS

Task Name	Task Number	Starting Address
UPDATE (Update time)	Ø	5Ø2Ø
ICAHND (ICA handler)	1	7Ø2Ø
SICHND (SIC handler)	2	782Ø
SMHND (SM handler)	3	682Ø
CMDITR (Command Interpreter)	4	6Ø2Ø
NRTSK (Non-resident task)	5	Ø4ØD (usually)

The Update task works in conjunction with a timer interrupt service routine. The timer interrupt routine increments a counter with each clock tick. When the Update task runs it reads this tick counter, updates time of day, and decrements each task's delay counters accordingly. When a task's delay count reaches zero, the corresponding task is activated.

The Shared Memory (SM), Interface Configuration Adaptor (ICA), and Nameplate (NP) handlers perform communication and data transfers with their respective devices. Each handler also updates its device's status in Shared Memory. The handlers run as tasks, and are activated by other tasks through executive requests.

The Command Interpreter task works in conjunction with an LM function interrupt routine which runs whenever the Link Manager (LMG) sends a command to the LM via Shared Memory. The interrupt routine checks the command for validity and flags the Command Interpreter task to execute the command.

A non-resident task may be loaded and executed in the LM. This program may be either uploaded from the Nameplate or downloaded from the LMG. It may be a data I/O, calibration, or subsystem diagnostic. However, only one non-resident task may be loaded in the LM at any one time. This task may be started or stopped under LMG control.

Tasks are scheduled for execution under a round robin scheduling algorithm. Details of the executive and each task are given in the sections that follow.

2.3.2 REAL-TIME EXECUTIVE

A real-time executive program is used to schedule the execution of tasks in the Motorola M6800 microprocessor based system. This section describes the services available in the executive and defines how to use them. A description of the functional aspects of the program is also given here.

The executive implements a round robin scheme for task scheduling. The executive allows for up to 12 tasks to be scheduled, including non-resident tasks. In this implementation it permits only 1 non-resident task and includes only 5 predefined resident tasks. The 5 resident tasks are: the update task, the Shared Memory (SM) handler, the Subsystem Information Channel (SIC) handler, the Interface Configuration Adaptor (ICA) handler, and the command interpreter. The non-resident task can be any one of several data I/O and diagnostic tasks. This constraint can easily be modified, if necessary, to include more non-resident tasks.

The Executive program allows each installed task to be in one of four possible states: dormant, delayed, ready, or running. The task states are diagrammed in Figure 13 . A running task is the one currently using the CPU. A task in the ready state is waiting for its turn to be processed. A delayed task becomes ready when its delay time has elapsed. A dormant task will not run. A delayed or dormant task may be brought to the ready state through activation.

Ready tasks are executed in a cyclic manner, the next ready task in the cycle being given control of the CPU. Once a task has control of the CPU, it is up to it to voluntarily release the CPU. Therefore, each task must not be executed continuously if it requires excessive CPU time, otherwise the system may not maintain a real-time operation.

A running task can call upon the executive to modify the execution status of another task - (activate, abort, install or remove) in which

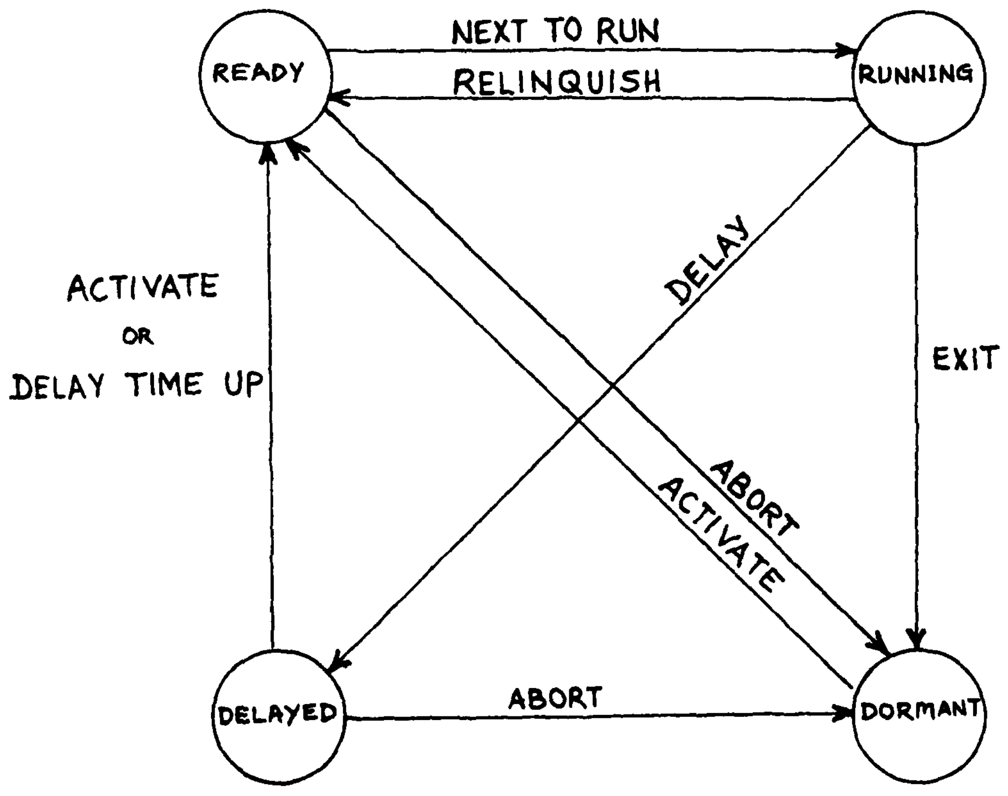


Figure 13 Transition of States

case the executive temporarily regains control of the CPU, performs the function, and returns CPU control to the calling task. A running task can release CPU control by calling upon the executive to perform any of the following functions - relinquish, exit, or delay, in which case the executive performs the function on the calling task and transfers CPU control to the next ready task in the cycle. To be able to perform all the above mentioned functions, the executive maintains tables containing the start address, restart address, initial stack pointer, current stack pointer and initial and current state of task for each installed task.

In addition to task control services, the executive also provides services to allocate the three device handlers to tasks in need of them. This ensures that not more than one task is using a particular handler at any given time. The services are called Shared Memory Request (SMRQ), SIC Request (SICRQ), and ICA Request (ICARQ).

The executive also furnishes math functions which can be used by the running task. The functions provided are 16 bit divide, 16 bit multiply, binary to BCD, BCD to binary, and three special functions pertinent to the processing required by the subsystems used in this demonstration. The running task can call upon the executive to perform these functions. Upon completion, the executive returns to the calling task with the result.

Thus a total of 11 executive services are provided. The calling sequence and a brief description of each service is described below.

2.3.2.1 Executive Services

There are 11 executive services. Each service can be requested through the EXECRQ macro. This macro simplifies the calling sequence of all the services. The macro is listed and explained in Appendix C, Section 2-B.

The explanation of the executive services is given below.

1. Activate: This service is used by the running task to bring any other task into the ready state from either the dormant or delayed states. It has no effect on a task already in the ready state.

   Calling sequence: EXECRQ ACTVAT, TASKNO

   where TASKNO is the task # of task to be activated

   Register usage: A - #0

   B - #TASKNO

   X - unused

2. Relinquish: This service transfers the running task from the running state to the ready state. This is one of the ways a task can release CPU control, and ensures the task's regaining CPU control after one cycle of the round-robin scheduler, whereupon it can start wherever it last left off.

   Calling sequence: EXECRQ RELQSH

   Register usage: A - #1

   B - unused

   X - unused

3. Exit: This service transfers the running task from the running state to the dormant state. This also is a way for a task to release CPU control, but the task runs again only if activated by another task. Execution of a task that has exited starts from the beginning.

   Calling sequence: EXECRQ EXIT

Register usage: A - # 2

B - unused

X - unused

4. Delay: This service transfers a running task to the delayed state. This is another way for a task to release CPU control. The task will automatically become ready only when the delay time expires, unless prematurely activated by another task. When the task runs again it will start from wherever it last left off. The delay time is determined by the UPDATE task.

Calling sequence: EXECRQ DELAY, TIME

where TIME is the delay time in seconds if the MSB of the 8 bits = 1 and is the delay time in 10's *of milliseconds if the MSB of 8 bits = Ø.*

Register usage: A - #3

B - # TIME

X - unused

5. Abort: The running task can use this service to transfer any other task from the ready or delayed states to the dormant state. The aborted task will run again when activated, and execution will start from the beginning.

Calling sequence: EXECRQ ABORT, TASKNO

where TASKNO is the task # of the task to be aborted

Register usage: A - #4

B - #TASKNO

X - unused

6. <u>Install</u>: The running task can use this service to introduce a new task into the execution scheduling cycle. At present only one task can be installed and will be designated as task #5. Thus if there is already a task designated as #5 it will have to be 'removed' before 'installing' a new one. The service returns a status indicating the absence or presence of a previously installed task.

To install a new task it is necessary to transfer to the executive the starting address, initial stack pointer and initial state of the task.

Calling sequence: set up table as follows for task to be installed.

TABADR →	Starting address	2	
	Initial stack pointer	2	bytes
	Initial state	1	
	Return status	1	

```
. EXECRQ INSTAL, TASKNO, TABADR
```

where TASKNO is # of the task to be installed (5) and TABADR is the address of the start of the table set up.

Register usage:	A - #5
	B - #TASKNO
	C - #TABADR

7. <u>Remove</u>: This service is used by the running task to remove a task from the round-robin cycle. The task will be removed only if it is in the delayed or dormant state. If it is in the ready state, the

task will not be removed and a status will be returned indicating that the particular task was active.

Calling sequence: EXECRQ REMOVE, TASKNO, STSADR

where TASKNO is the # of the task to be removed and STSADR is the address where the status will be returned.

Register usage:	A - #6
	B - #TASKNO
	X - #STSADR

8. ICA Request: The running task can use this service to assume control of the ICA handler. If the services of the ICA handler have already been granted to another task the service call returns with a status saying so. If the handler is successfully allocated to the running task, the address of the table of parameters to be sent to the handler is passed to the handler. Then, status saying 'handler allocated' is returned to the calling task.

Calling sequence: EXECRQ ICARQ, ICAFUN, GROUP, CHANEL, OPTION, CNSORC, NUMBYT, BUFADR, UFTADR

where UFTADR is the address where the parameter table is to be set up. A detailed description of other parameters will be found in the ICA handler section.

Register usage:	A - #7
	B - unused
	X - #UFT addr.

9. SIC Request: The running task can use this service to assume control of the SIC handler. This service functions exactly like the ICA request service.

Calling sequence: EXECRQ SICRQ, NPID, SICFUN, BUFADR, BUFSIZ, UFTADR

where UFTADR is the starting address of the table of parameters. For a description of other parameters please refer to the SIC handler section.

Register usage:	A - #8
	B - unused
	X - #UFTADR

10. SM Request: The running task can use this service to assume control of the shared memory handler. This service functions exactly like the ICA request service.

Calling sequence: EXECRQ SMRQ, SMFUN, BUFADR, BUFSIZ, UFTADR

where UFTADR is the starting address of the table of parameters. For a description of other parameters please refer to the SM handler section.

Register usage:	A - #9
	B - unused
	X - #UFTADR

11. Math: The running task can use this service to perform any one of seven mathematical functions. The task must provide the arguments in the calling sequence. The executive returns to the calling task

with the result of the operation in the address specified by the calling task.

General calling sequence:

```
EXECRQ MATH, function, ADROP1, ADROP2, ADROP3, ADROP4, RESULT
```

where function is any one of:

DMULT	- Double multiply (8 bit operands, 16 bit result)
DDIV	- Double divide (16 bit operands, 16 bit result)
BINBCD	- Binary to BCD (16 bit binary to 5 BCD digits)
BCDBIN	- BCD to binary (4 BCD digits to 16 bit binary)
CALCA	- Calculation of synchro constant 'A'. Arguments required are three synchro voltages each in 8 bit 2's complement form.
THETA	- Calculation of synchro angle 'θ'. Arguments required are three synchro voltages each in 8 bit 2's complement form and value of 'A'.
VOUTS	- Calculation of three synchro output voltages in 8 bit 2's complement form. Arguments required are 'θ' and 'A'.

ADROPn is address of operand n

and RESULT is address for the result.

Calling sequences according to functions:

```
EXECRQ MATH,DMULT,addr. of multiplicand,addr. of multiplier,,,RESULT
EXECRQ MATH,DDIV,addr. of dividend,addr. of divisor,,,RESULT
EXECRQ MATH,BCDBIN,addr. of BCD,,,,RESULT
EXECRQ MATH,BINBCD,addr. of binary,,,,RESULT
EXECRQ MATH,CALCA,addr. of V1,addr. of V2,addr. of V3,,RESULT
EXECRQ MATH,THETA,addr. of V1,addr. of V2,addr. of V3,addr. of A,RESULT
EXECRQ MATH,VOUTS,addr. of θ,addr. of A,,,RESULT
```

2.3.2.2 Operation

To be able to function properly, the executive needs to keep track of three parameters for each task scheduled for execution. It must know the state (active/inactive) of the task, the address at which the task will resume execution, and the value of the stack pointer when the task resumes execution. All these parameters may vary during the course of execution of a task, but initially - before the task starts its first execution - they will always have a fixed value chosen during system generation. So the initial values of these parameters are stored in ROM and during its initialization the executive copies these parameters into RAM. From thereon, the executive examines and/or modifies these parameters in RAM during context switching.

The executive also needs to know the number of tasks installed. This number is stored in a variable called NTASKS, which varies due to the 'installation' or 'removal' of the nonresident task. The variable is initialized by the executive during its initialization. Since in this application there are 5 resident tasks and no initially installed nonresident task, this variable is first set to 5.

Thus we see that during its initialization the executive copies parameters from ROM into RAM and sets-up the number of tasks. It also allows the 5 resident tasks to go through their own initialization. All this is done on power-up.

Once this is over, the scheduler part of the executive takes over. It determines which task should get control of the CPU by examining the task status of each task. The first task it comes across which is in the active state receives CPU control. Before transferring control, the scheduler gets the value of the stack pointer and the address where the task will begin execution (both from RAM). It then sets up the stack pointer and jumps to the start address obtained from RAM.

Whenever a task wants to avail itself of the executive's services, it will make a call to the executive - through the EXECRQ macro - at which point the service dispatcher part of the executive will take over. The service dispatcher will examine the contents of register A and will jump to the appropriate service. The particular service will be performed and then one of two things will take place. If the service is either relinquish, delay or exit, control of the CPU will go to the scheduler so that the next active task may be scheduled for execution. On the other hand if the call is for a service other than the three mentioned, the executive will return control to the calling task.

The architecture of the executive is portrayed in Figure 14. A list of routines used by the executive and their respective functions, is presented in Table 3.

2.3.2.3 Parameters and Variables

The executive requires certain parameters and maintains various variables (some of which are global) to enable its functioning. These variables give a fair indication of the state of the executive and the resident and nonresident tasks, and their examination is a useful debugging aid. A brief explanation of the most significant parameters and variables will now be presented.

Parameters

The task parameters that are copied from ROM into RAM during the exe-

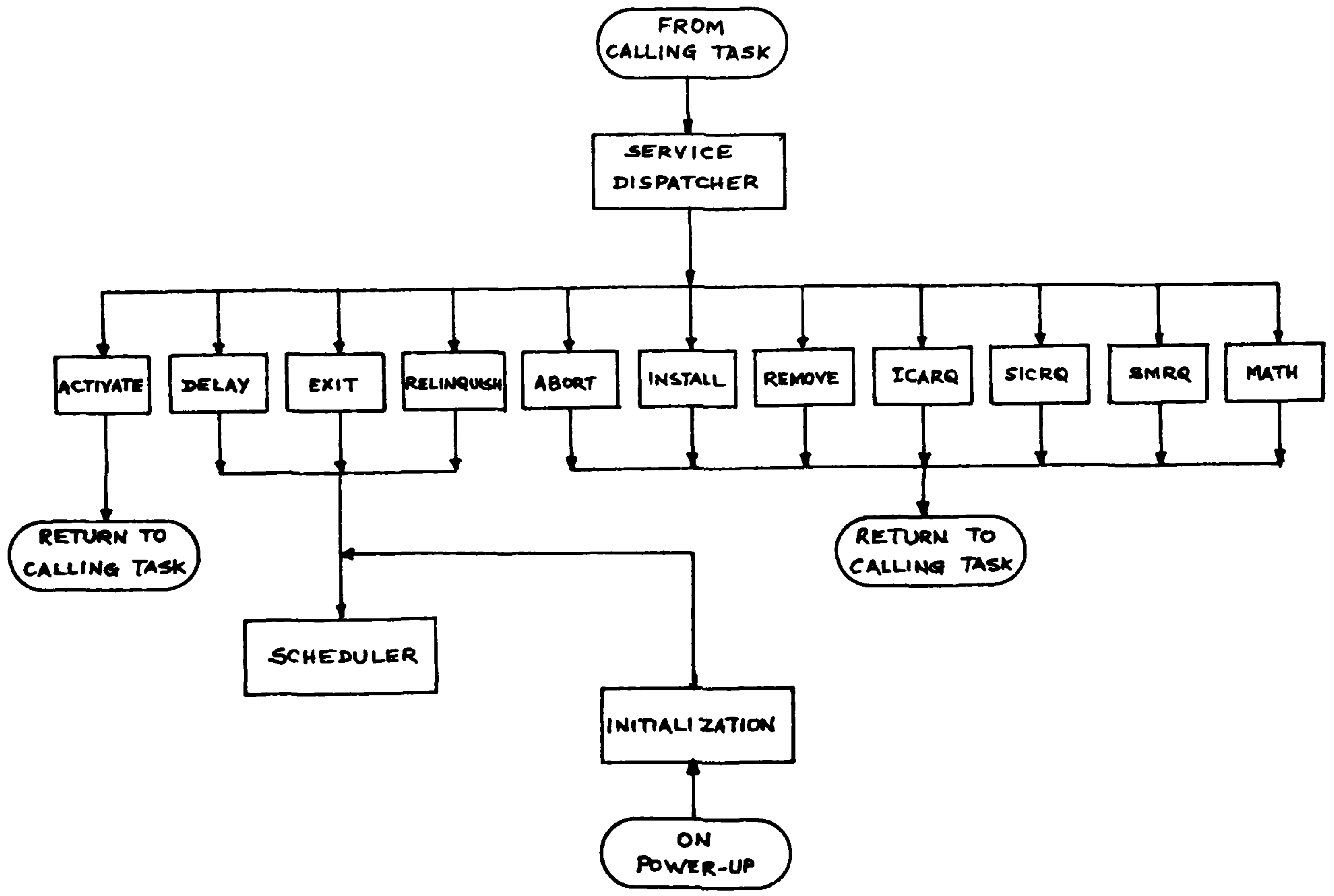


Figure 14 The Real-time Executive and Services

Table 3

PROGRAMS AND THEIR FUNCTIONS

PROGRAM NAME	DESCRIPTION
EXEC	Main program
ACTVT	Service routine for ACTIVATE function
ABORT	Service routine for ABORT function
DELAY	Service routine for DELAY function
EXIT	Service routine for EXIT function
RELQSH	Service routine for RELINQUISH function
INSTAL	Service routine for INSTALL function
REMOVE	Service routine for REMOVE function
ICARQ	Service routine for ICA handler allocation
NPRQ	Service routine for SIC handler allocation
SMRQ	Service routine for SM handler allocation
MATH	Service routine for MATH operations
BLDADR	Adds the contents of Reg. A to Reg. X
CLRACT,CLRACI	Clears the active bit of task status word
GETSTS	Gets the status word for a task
GETADR	Performs the operation: Reg(X)=2. Reg(A)+Reg(X)
GSTADR	Transfers task start address to restart address
ADJSTK	Adjusts task stack pointer to its initial position
COPY2B	Copies two bytes of data
INITIA	Routine which initializes the EXEC and each task
XFER	Subroutine that transfers a block of N bytes from loc. 1 to loc. 2
RESTOR	Copies start addr. + initial SP. for nonresident task
DMULT	Does a double multiply
DDIV	Does a double divide
BCDBIN	Converts 4 BCD's to 2 byte binary
BINBCD	Converts 2 byte binary to 5 BCD's
CALCA	Calculates 'A' from synchro voltages
THETA	Calculates θ from synchro voltages and 'A'
VOUTS	Calculates synchro output voltages from θ and 'A'
SERCH	Binary search routine
COSINE	Determines Cos θ

cutives initialization are the following:

STAADR (Task starting address array) - Two bytes for each resident task giving the address where the task first starts execution after initialization.

STKROM (Task stack pointer array) - Two bytes for each resident task containing the initial value of the stack pointer for the task.

INIZST (Task initial status array) - One byte for each resident task specifying the initial state of the task at system start-up.

Global Variables

The following variables are used by the executive:

TSKSTS (Task status array) - Initially copied from INIZST. One byte for each task indicating if the task is active ($8Ø) or not ($ØØ). *Also* accessed by the update task. Upon expiration of the delay time of a delayed task, the update task changes the task's status from inactive to active. The executive examines this array for scheduling of tasks.

DLYTIM (Task delay times) - One byte for each task indicating the delay time of the corresponding task. The most significant bit shows the delay time is in seconds (1) or 10's of milliseconds (Ø). The other 7 bits are a delay unit count. The executive initializes the delay time as requested by the calling task and the update task modifies it as time ticks.

MATHVAR (P,Q,R,TABLE) - P, Q and R are 2 bytes each and TABLE is 6 bytes. The math routines in the executive utilize these variables during computation. These are initialized - with the operands and address of the result - by the macro whenever a task requests the math service of the executive.

UFTICA (UFT* pointer for ICA handler) - Two bytes containing the 16 bit address of the start of the UFT table holding parameters to be sent to the ICA handler. This is initialized by the ICA handler during its initialization.

UFTSM (UFT pointer for SM handler) - Two bytes containing the 16 bit address of the start of the UFT table holding the parameters to be sent to the SM handler. This is initialized by the SM handler during its initialization.

UFTNP (UFT pointer for SIC handler) - Two bytes containing the 16 bit address of the start of the UFT table holding the parameters to be sent to the SIC handler. This is initialized by the SIC handler during the initialization.

INSFLG ('Installed' flag) - One byte indicating whether a nonresident task is installed ($Ø1) or not ($ØØ). It is also examined by the command interpreter.

A list of global variables appears in Table 4. The interactions of the executive with other tasks is portrayed in Figure 15.

Other Variables

RSTADR (Task restart address array) - This is initially copied from STAADR. It has two words for each task containing the address at which the corresponding task next resumes execution.

*User File Table

Table 4

LIST OF GLOBAL COMMON VARIABLES

NAME	IN/OUT	SIZE	TYPE	DESCRIPTION
TSKSTS	Both	12	1-bit flag	Task status word. One word for each task. Leading bit shows a task is ready (1) or not (Ø).
DLYTIM	Out	12	7-bit binary	Task delay timer. One word for each task. The most significant bit shows delay time in seconds (1) or 10's of mS (Ø). Other 7 bits are a delay unit count.
UFTICA	Out	2	binary	2 bytes for passing UFT address to the ICA handler.
UFTNP	Out	2	binary	2 bytes for passing UFT address to the SIC handler.
UFTSM	Out	2	binary	2 bytes for passing UFT address to the SM handler.
INSFLG	Both	1	1-bit flag	If nonresident task installed (1) or not (Ø) indicated by LSB.
MATHVAR (P,Q,R, TABLE)	Both	12	binary	Variables used by MATH routines holding the operands (P,Q,TABLE) and address where result is to be returned (R).

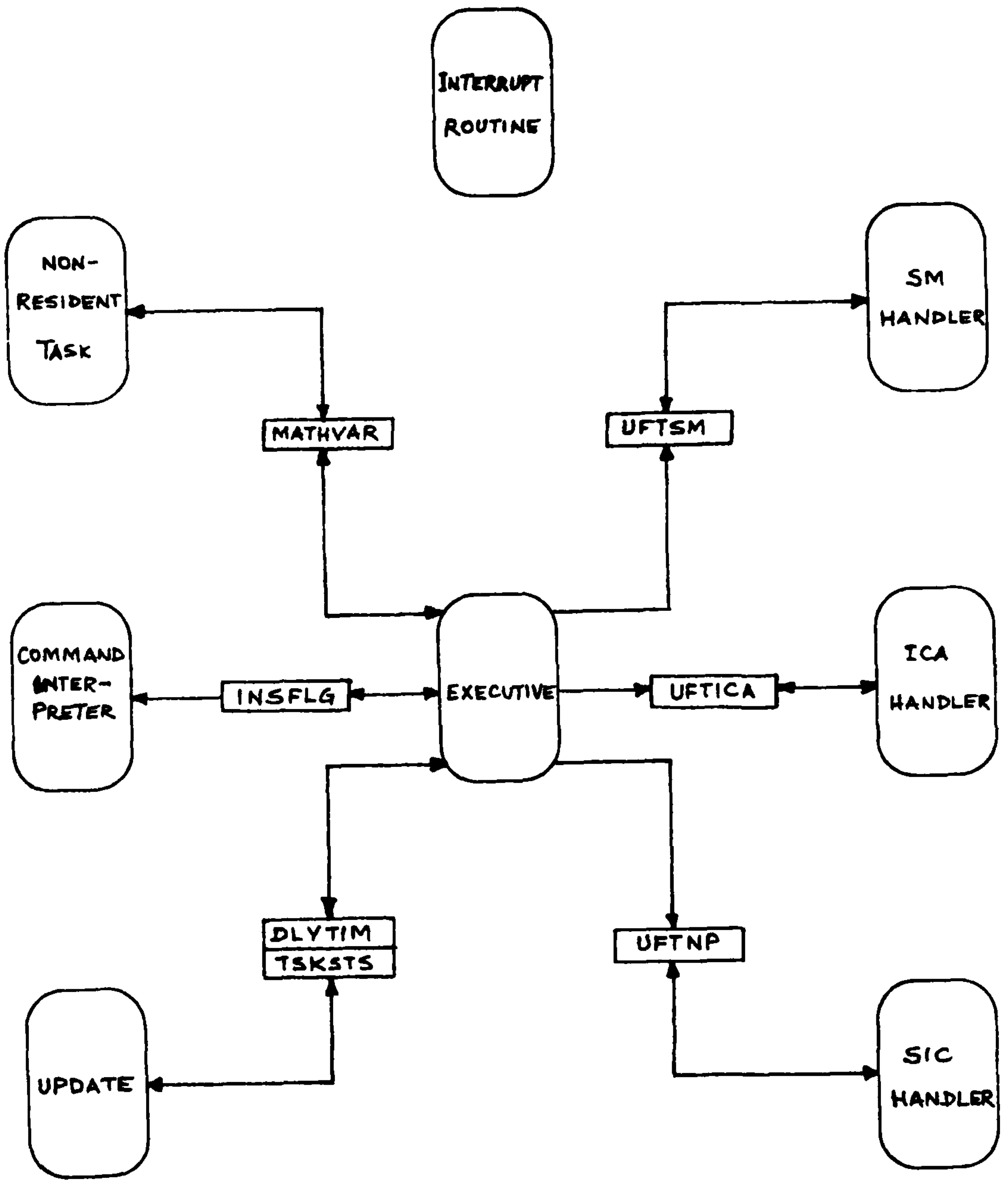


Figure 15 Interaction of the Executive

STKRAM (Task stack pointer array) - This is initially copied from STKROM. It has two words for each task containing the current value of the stack pointer for the task.

RSTTMP - Two words for the nonresident task containing the address at which it will first start execution. This is analogous to STAADR for resident tasks.

STKTMP - Two words for the nonresident task containing the initial value of its stack pointer. This is analogous to STKROM for resident tasks.

NTASKS - One byte indicating the number of tasks in the round-robin, including nonresident tasks if any. This is initialized to 5 and changed to 6 if a nonresident task is installed.

There are other variables used by the executive but since their description does not in any way shed light upon the functioning of the executive, their explanation has been omitted.

2.3.3 UPDATE TASK

The real-time executive used in this system requires a task to keep track of time. The executive also needs the facility of updating the status of delayed tasks. The update task ostensibly fulfills these requirements, with the help of some hardware and an interrupt routine.

The update task maintains time in Julian day, hours, minutes, and seconds. It decrements the delay times in the delay-time array of the executive. It also maintains 6 independent timers for the handlers.

2.3.3.1 Operation

In order for it to count time, the update task must have access to a hardware clock. Each clock pulse causes an interrupt which is serviced by an interrupt routine called CLOCK. The interrupt routine accumulates the number of pulses received from a 100 Hz clock.

The update task is scheduled for execution in the same way as other tasks. Each time the update task runs, it checks to see if any pulses have been accumulated by the interrupt routine. It decrements the delay times of all delayed tasks by the amount of time the number of pulses add up to. It checks the resulting delay time, and if the time has expired it changes the state of the corresponding task from 'delayed' to 'ready'. Figure 16 shows the structure of the update task.

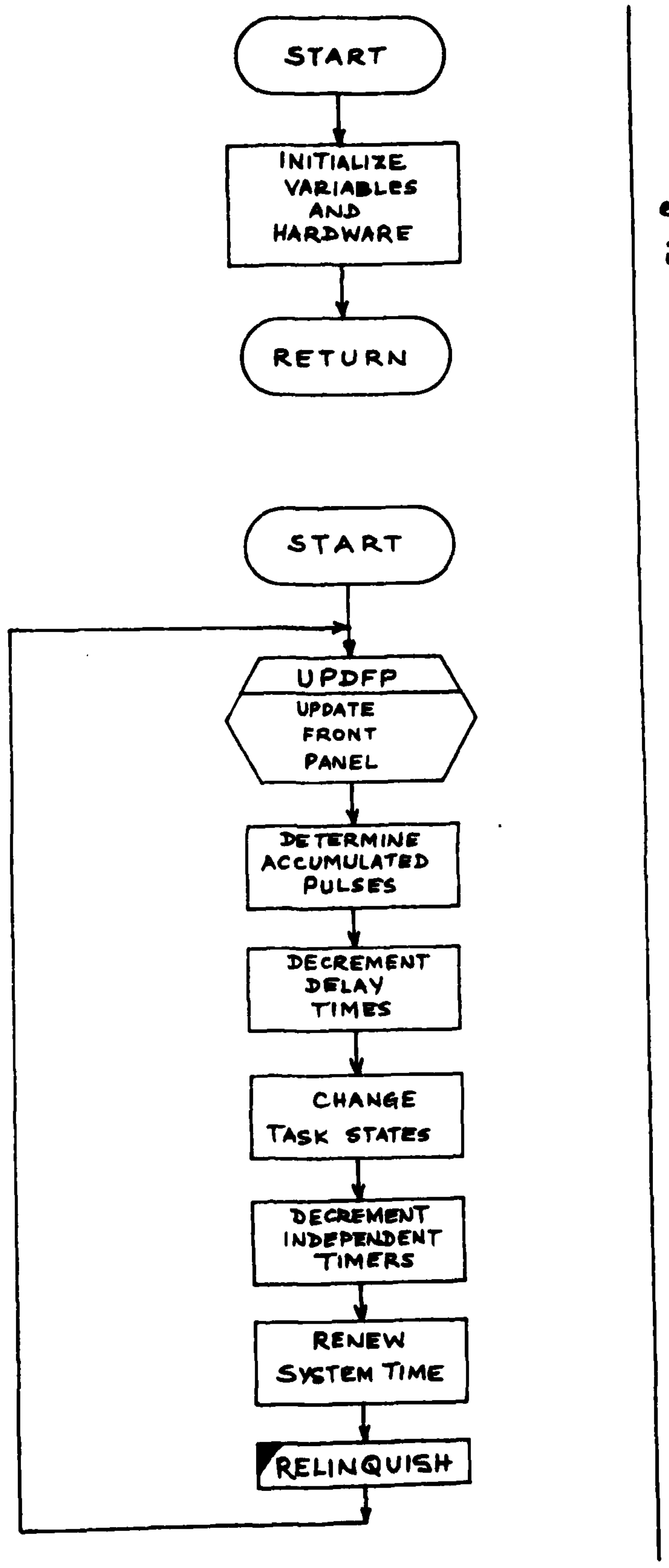
Next, the update task decrements the 6 independent timers used by the handlers. These timers are in units of 10 milliseconds and thus allow a maximum time count of 2.55 seconds.

Lastly, the update task renews the system time. The time - in days, hours, minutes, and seconds - is maintained in BCD data format.

The task then relinquishes control of the CPU and awaits for its turn in the next scheduling cycle. In the next cycle it goes through the above mentioned steps in the same fashion.

Since the update task runs during every scheduling cycle, it is used to call upon a subroutine which refreshes the front-panel of the system.

A detailed description of the update task is presented in Appendix C, Section 2-C.



COMMENTS

Invoked by executive during its initialization.

Figure 16 Update Task Structure

2.3.3.2 Interactions with other Tasks

The update task has interations with the executive, the handlers, the interrupt service routine, the command interpreter, and the nonresident tasks. These interactions are shown in Figure 17 along with the names of the variables through which they take place. For a detailed description of these variables, refer to Appendix C, Section 2-C.

2.3.4 ICA HANDLER

The ICA handler is one of the resident tasks in the link module and is designated as task #1. It is a software program which provides an interface between a task wanting to use the ICA and the ICA itself. It translates general commands from a task, into specific commands recognizable by the ICA. It also provides a means of transfer of data and status between a task and the ICA, taking care of all the details involved with this transfer. For a task, this implies a simplified means of interacting with the ICA.

The handler communicates with the ICA through the ICA buffers. A map of these buffers is given in Figure 18. As shown in this figure, some of these buffers are used to configure the ICA, and others are used for transferring data.

2.3.4.1 Handler Functions

The ICA handler has 3 basic functions:

1. Configure
2. Control
3. Data transfer

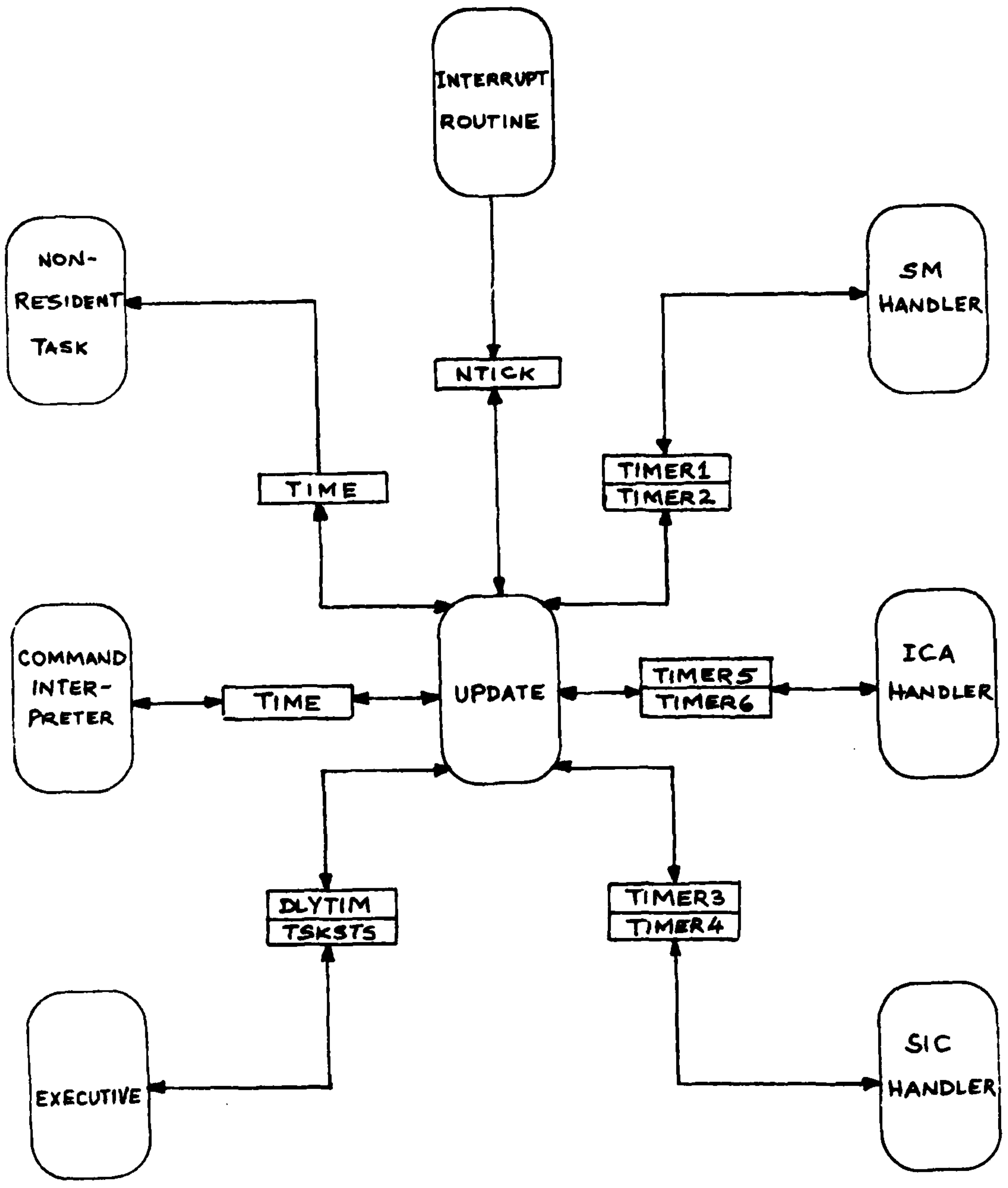


Figure 17 Update Task Interactions

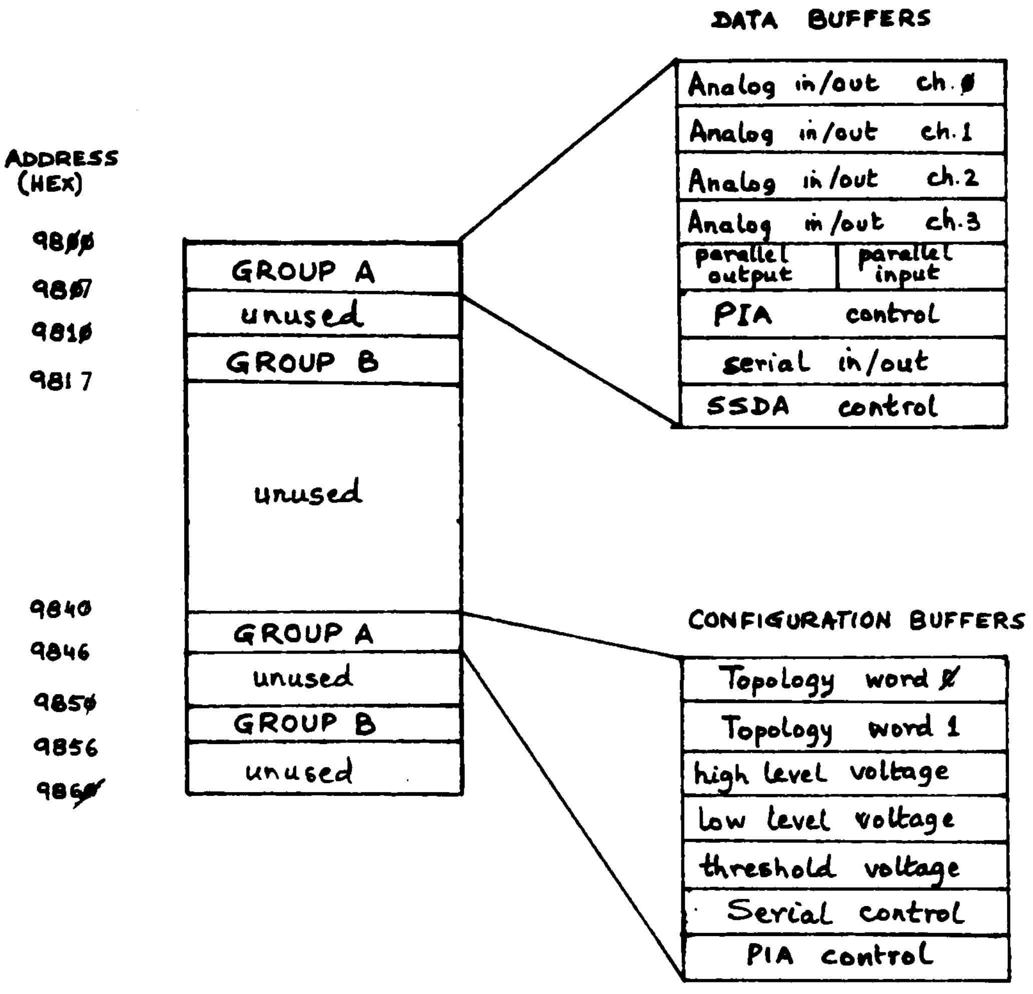


Figure 18 Memory Map of ICA Buffers

A brief description of each of these is given below.

Configure: This function has two options - read and write. If the option is 'Read', the handler transfers the existing configuration to a specified buffer address. If the option is 'Write', the handler transfers the desired configuration from a specified address to the ICA buffers.

The ICA has several configurations, each of which defines a particular combination of states of its hardware. Any configuration consists of 7 bytes of data. The handler uses this data to configure the ICA. A list of configurations is given in Table 5.

Control: This function has 3 options - online, offline and reset. 'Online' enables outputs from the ICA, 'Offline' disabled outputs from the ICA. 'Reset' disables both inputs and outputs.

Data Transfer: This function has 2 options - input and output. The data must be transferred to or from the handler, in a buffer. The handler assumes that the data being transferred is compatible with the configuration of the ICA. However, if the option is input and the ICA is configured for output or vice-versa, the handler detects the error.

2.3.4.2 Calling Sequences

The services of the handler must be requested through the executive. The request to the executive is made through the following macro call:

```
EXECRQ ICARQ, ICAFUN, GROUP, CHANEL, OPTION, CNSORC, NUMBYT, BUFADR, UFTADR
```

EXECRQ is a macro defined to form a table of all these parameters. The table will start at the UFT address specified and will be set up as shown in Figure 19.

Table 5

ICA CONFIGURATIONS

TYPE	TOP. WORD Ø	TOP. WORD 1	HI LEVEL	LO LEVEL	THRSH	WORD COUNT	CLOCK RATE
AINAC S.E.	C3	30	4Ø	8Ø	6Ø	Ø2	00
AINAC DIFF.	C7	3Ø	4Ø	8Ø	6Ø	Ø2	ØØ
AINDC S.E.	83	3Ø	4Ø	8Ø	6Ø	Ø2	ØØ
AINDC DIFF.	87	3Ø	4Ø	8Ø	6Ø	Ø2	ØØ
AOUTAC S.E.	CB	52	4Ø	8Ø	6Ø	Ø2	ØØ
AOUTAC DIFF.	DF	52	4Ø	8Ø	6Ø	Ø2	ØØ
AOUTDC S.E.	8B	52	4Ø	8Ø	6Ø	Ø2	ØØ
AOUTDC DIFF.	9F	52	4Ø	8Ø	6Ø	Ø2	ØØ
SYNIN	CB	1Ø	4Ø	8Ø	6Ø	Ø2	ØØ
SYNOUT	CB	2Ø	4Ø	8Ø	6Ø	Ø2	ØØ
SINREF	ØB	42	4Ø	8Ø	6Ø	Ø2	ØØ
SINFLG	ØB	46	4Ø	8Ø	6Ø	Ø2	ØØ
SOUT	ØB	82	4Ø	8Ø	6Ø	Ø1	ØØ
DINREF S.E.	Ø3	3Ø	4Ø	8Ø	6Ø	Ø2	ØØ
DINREF DIFF.	Ø7	3Ø	4Ø	8Ø	6Ø	Ø2	ØØ
DINMOM FOL.	23	32	4Ø	8Ø	6Ø	Ø2	ØØ
DINMOM LAT.	23	33	4Ø	8Ø	6Ø	Ø2	ØØ
DOUT	ØB	52	4Ø	8Ø	6Ø	Ø2	ØØ

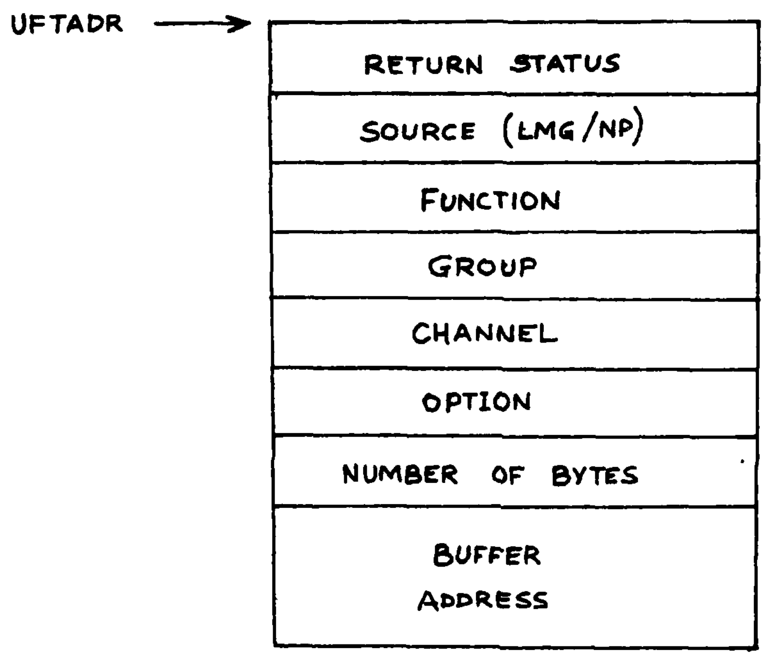


Figure 19 Parameter Table Set-up by Macro

If the executive grants the request for the handler, then it saves the pointer to this table - the UFT address. When the ICA handler runs the next time, it retrieves the UFT address and parameters, and performs the function as specified by the parameters.

An explanation of the parameters in the calling sequence is given below.

ICAFUN - This parameter specifies the function. It could be CONFIG, CONTRL, or DATATR.

GROUP - This is used only when ICAFUN is DATATR, and when the data being transferred is analog. It specifies the channel and could be 1, 2, 3, 4, or 5 (for all channels).

OPTION - Every function has certain options. When ICAFUN is CONFIG, possible options are READ and WRITE and are used, respectively, to read or write the configuration. When ICAFUN is CONTRL the possible options are RESET, ONLINE, and OFFLINE. RESET resets the ICA, ONLINE enables outputs, and OFFLINE disables them.

When ICAFUN is DATATR the possible options are INPUT and OUTPUT and are used, respectively, to input and output data.

CNSORC - This parameter defines the source of the configuration. If the configuration is from the LMG this parameter is 'Ø' and if the configuration is from the nameplate this parameter is '1'.

NUMBYT - This parameter holds the number of bytes either to be transferred or that have been transferred. It is not necessary to supply the number of bytes to be transferred since in all cases this is known apriori. However, this parameter does hold the number of bytes successfully transferred, after the transfer takes place. If this parameter is zero it indicates that the transfer was unsuccessful.

BUFADR - This parameter holds the address of the 7 bytes of configuration that will be used if ICAFUN is CONFIG and the option is WRITE. For the same function if the option is READ, this parameter holds the address of the buffer into which the configuration will be read.

If ICAFUN is DATATR then BUFADR holds the address of the buffer to or from where the data will be transferred, depending on the OPTION.

If ICAFUN is CONTRL, this parameter is not used.

In the table of parameters as shown in Figure 19, the first entry is 'Return status'. This holds the status of the operation as returned by the handler. The various status codes and their meaning is given in Table 6.

2.3.4.3 ICA Handler Design

The ICA handler has an initialization section which is invoked by the executive during the executive's initialization. During initialization the handler sets-up the Peripheral Interface Adapters (PIA) and the syn-

Table 6

ICA HANDLER STATUS CODES

Status	Description
Ø	Success
1	Handler allocated.
2	Handler not allocated.
4	Function in progress.
-1	Invalid function.
-2	Group not configured.
-3	Parity error during serial input.
-4	Invalid configuration.
-5	No data received during serial input.
-6	Parity error during serial output.
-7	Data not transmit during serial output.
-8	Invalid I/O request: group set for synchro output.
-9	Invalid I/O request: group set for synchro input.
-1Ø	Invalid I/O request: group set for analog output.
-11	Invalid I/O request: group set for analog input.
-12	Invalid I/O request: group set for discrete output.
-13	Invalid I/O request: group set for discrete input.
-14	Invalid I/O request: group set for serial output.
-15	Invalid I/O request: group set for serial input.

chronous serial data adapters (SSDA) which form part of the ICA buffers. The handler initializes all its variables and sets the ICA in the reset state. It also initializes the UFT pointer, UFTICA, to a parameter table which will require the handler to perform the update function. The update function, though not a 'full-fledged' function of the handler, updates the ICA status in shared memory.

Once the initialization is over, the handler runs periodically every one second. Whenever another task requests the handler's services this period is disrupted since the handler immediately attends to the request. Each time the handler runs, it first gets the parameter table pointer from UFTICA. From the parameter table it gets the function and performs the required steps corresponding to that function.

After performing any function, the handler sets UFTICA to point to the parameter table holding the update function. Thus, until the handler is requested by another task it periodically performs the update function. This structure of the handler is depicted in Figure 20.

The three major functions of the handler are implemented as subroutines. Each of these subroutines, in turn, call upon a host of other subroutines at the secondary level. The subroutines in the handler and their hierarchy are shown in Figure 21. For a detailed description of these subroutines please refer to Appendix C, Section 2-D.

The handler has interactions with the executive through UFTICA, with the update task through the independent timers TIMER5 and TIMER6, and with shared memory through ICASTS. These interactions are shown in Figures 15 , 17, and the shared memory map in Figure 28, respectively. For further details about these global variables please refer to Appendix C, Section 2-D.

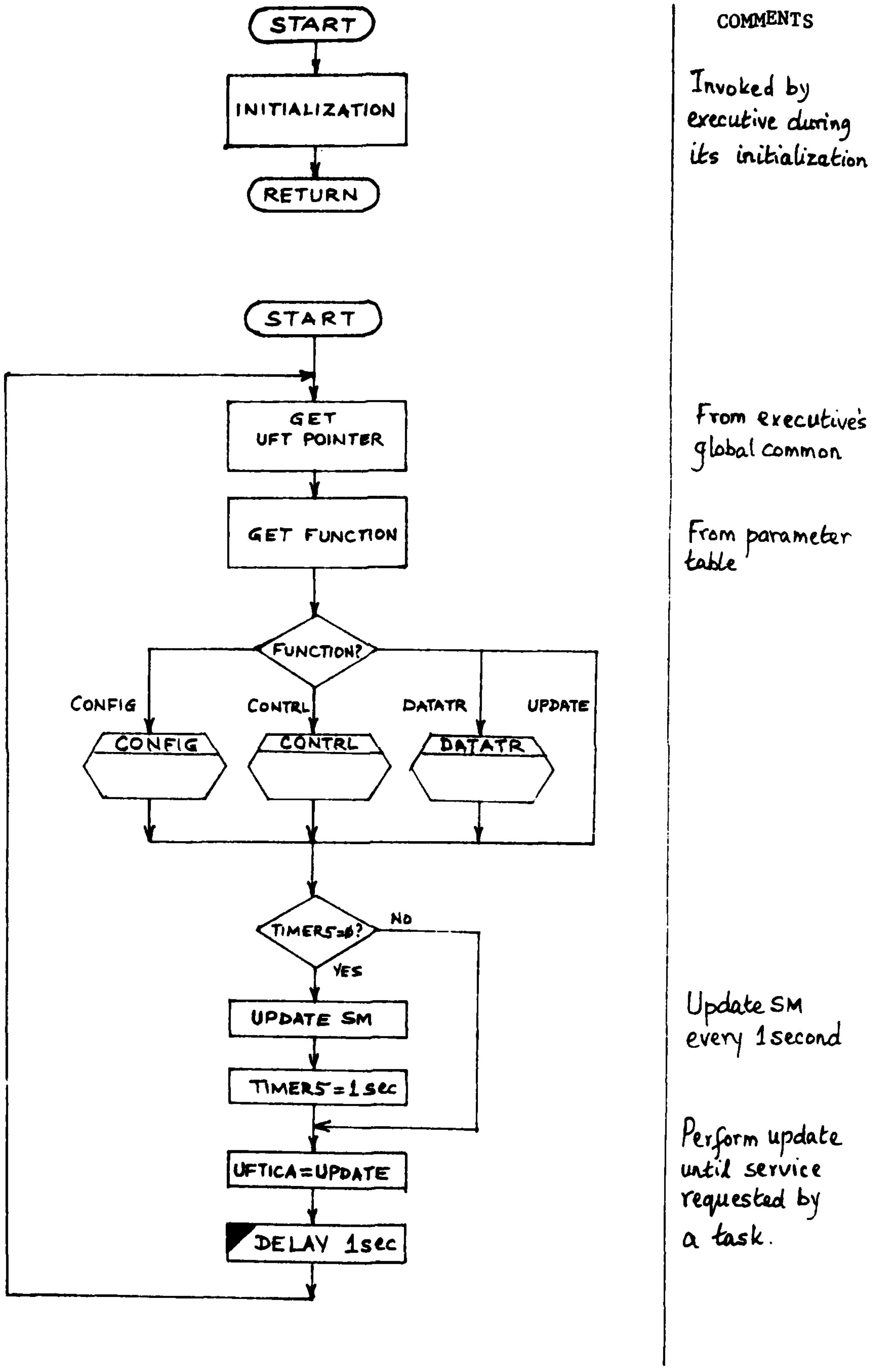


Figure 20 ICA Handler Structure

MAIN	SECOND LEVEL	THIRD LEVEL	FOURTH LEVEL	GENERAL PURPOSE
ICAH1				XFER
CONFIG	VALCHK			RETSTS
	PIASET	SIMSET	BUFSET	INCR
		SYNSET		
	FLGSET			
	TYPCHK			
CONTRL				
DATATR	SERIN	SIMSIN		
		RXDATA		
		START		
		SERSTP		
	DSCTIN			
	ANALIN			
	SYNCIN			
	SEROUT	SOSIM		
		XMIT	INST	
		SERSTP		
	DSCOUT			
	ANAOUT			
	SYNOUT			

Figure 21 ICA Handler Subroutines and their Hierarchy

2.3.5 SIC HANDLER

The SIC handler provides tasks running in the link module with a simple to use interface to the subsystem information channel (SIC). The handler allows a calling task to make a single call to request a relatively complex function be performed. The SIC handler will translate a request into several commands which are sent to an electronic nameplate to perform the function requested. All the detailed timing and control required for communication with electronic nameplates are performed by the handler.

2.3.5.1 SIC Handler Functions

This section describes the functions of the SIC handler. Table 7 lists all SIC handler functions implemented.

2.3.5.1.1 SIC Status in Shared Memory

The SIC handler periodically updates a status byte in shared memory. This byte reflects the status of the subsystem information channel as shown in Figure 22. The handler updates this byte every 0.5 seconds.

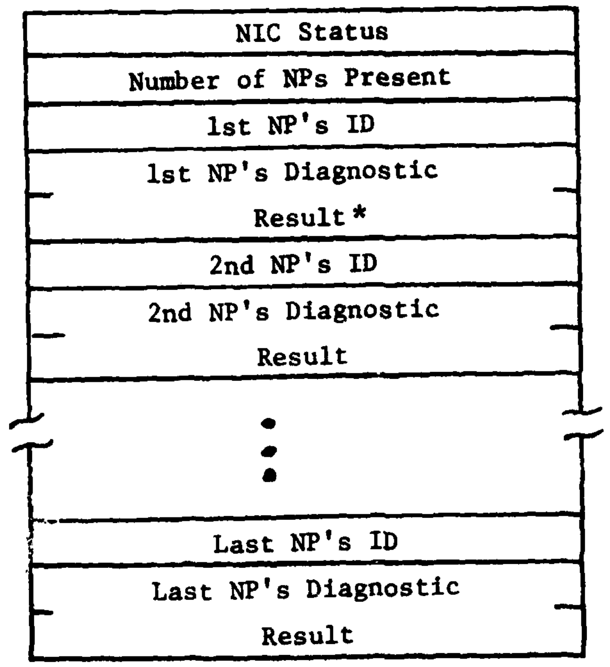
2.3.5.1.2 SIC Initialization

Any time a nameplate is added to or removed from the SIC, a request for initialization must be made to the handler. This is accomplished through the SIC handler function SICINT. This function request causes the handler to reset the nameplate, to run the nameplate's diagnostic program, and to build the SIC status table shown in Figure 23. This initialization procedure is the only way bits 7 and 6 (SIC hardware failure and SIC configuration changed) of the SIC status byte in shared memory may be reset after a SIC status condition has caused them to be set.

b7	b6	b5	b4	b3	b2	b1	bØ
HWE	CFC	NPP	INT	RCF	Ø	Ø	RST

HWE:	SIC hardware failure (master NP or NIC), used as No-Go/Go bit.
CFC:	SIC configuration changed since initialization (SIC needs to be reinitialized).
NPP:	At least one NP is present.
INT:	SIC has been initialized.
RCF:	SIC maintenance record area on the master NP is full.
RST:	SIC in reset state.

Figure 22 SIC Status Byte in Shared Memory


NIC Status
Number of NPs Present
1st NP's ID
1st NP's Diagnostic Result*
2nd NP's ID
2nd NP's Diagnostic Result
Last NP's ID
Last NP's Diagnostic Result

*See Section 4 for format of these diagnostic result bytes.

Figure 23 SIC Status Table

2.3.5.1.3 Load Functions

The load functions (LDDIR,LDCFN,LDCNV, and DMPWRT) cause different areas of a nameplate's memory to be read and stored in the area specified by the calling task. The type of data each function causes to be loaded is indicated in Table 7.

2.3.5.1.4 Maintenance Record Functions

A task running in the LM may write maintenance records regarding the subsystem's performance into the read/write area of a nameplate's memory. The SIC handler function WRTREC is used to perform this function. A maximum of 14 bytes (one record) may be written at a time. The format of a maintenance record is described in Section 4.1.3. Records are written sequentially in the nameplate's read/write memory.

The RDREC command is used to read a previously written record. A record pointer is used to specify the first record to be read. The function RDREC causes the specified number of records to be read starting with the record pointed to by the record pointer. The function RECPOS is used to move the record pointer to a desired record. There are four sub-functions (POSFUN) of the function RECPOS used for this pointer as indicated in Table 7.

The records may also be read by the load function, DMPWRT, which reads all 16 records (256 bytes total) from the nameplate at one time. The function ERAWRT simulates the erasure of the nameplate's read/write memory. The record pointer is set to point to the first record as a result of this erase function.

Table 7

SIC HANDLER FUNCTIONS

FUNCTION	CODE (SICFUN)	DESCRIPTION
UPDTSM	Ø	Update SIC status in shared memory.
SICINT	1	Initialize SIC.
LDDIR	2	Load nameplate's directory.
LDCFN	3	Load ICA configuration table from the nameplate.
LDCNV	4	Load the subsystem's data I/O conversion program from the nameplate.
	5	Reserved.
	6	Reserved.
WRTREC	7	Write a record into the nameplate.
DMPWRT	8	Load all of the nameplate's read/write memory (record area).
ERAWRT	9	Simulate erasure of the nameplate's read/write memory.
RECPOS	1Ø	Positions the record pointer to a particular record. The type of positioning (POSFUN) which may be requested are: • EDD: (POSFUN=1) end of data; positions record pointer to the first record following the end of written records (i.e. the first empty record) Note: this is where the record pointer is set after a record has just been written. The record pointer value is ignored in writing records. Records are always written sequentially.

Table 7 (Continued)

SIC HANDLER FUNCTIONS

FUNCTION	CODE (SICFUN)	DESCRIPTION
		• BOD: (POSFUN=2) beginning of data; positions the record pointer to the first record in the read/write memory. • BACKREC: (POSFUN=3) backup the record pointer the specified number of records. • FUDREL: (POSFUN=4) advance the record pointer the specified number of records.
RDREC	11	Read the specified number of records (NUMREC) from the nameplate starting at the record denoted by the record pointer.

2.3.5.2 SIC Handler Calling Sequence

A call to this handler by a task requesting a function is made by a request to the executive. The executive builds a user file table (UFT), consisting of the parameters specified by the calling task, and passes it to the SIC handler. Then the executive sets the SIC handler to a ready state, eliminating any remaining delay the handler had, so that the handler will run when its turn in the round robin schedule is reached. Note, however, that control is returned immediately to the calling task after the SIC handler is set ready. Thus the calling task must relinquish control to allow the handler to run.

The format of a request to the executive for this SIC handler is:

```
EXECRQ SICRQ, NPID, SICFUN, BUFADR, BUFSIZ, UFTADR
```

where

EXECRQ:	indicates a request of the executive is required
SICRQ :	indicates the SIC handler is requested
NPID :	specifies which nameplate is desired to perform function (must always equal zero in this implementation)
SICFUN:	specifies what function is requested of the SIC handler
BUFADR: (or POSFUN)	specifies the starting memory address of the memory area in the link module that data is to be loaded into (load functions) or copied out of (write record function). On a record positioning function this parameter becomes the type of record po•itioning required (POSFUN).
BUFSIZ: (or NUMREC)	specifies the maximum size of the memory area denoted by #BUFADR. On a record positioning function this parameter becomes the number of records to be processed. On the completion of any function this parameter is changed to represent the number of bytes actually transferred.
UFTADR:	specifies the memory area where the UFT is to be stored. The area size must be at least 8 bytes.

Figure 24 shows the UFT built as a result of this request.

Any time after a request is made to this handler, the calling task may examine the first word of the UFT to obtain the present status of the request. The request status codes are given in Table 8.

2.3.5.3 SIC Handler Design

Figure 25 shows the flowchart for the main routine of the SIC handler task SICHND. The SIC hardware and SIC internal variables are initialized through a subroutine, HNDINT, called by the executive as part of the link module power-up initializations. After this the SIC handler only executes the loop portion of the program shown in Figure 25.

When another task running in the LM requests of the executive an SIC handler function, the executive sets up an SIC handler user file table (UFT) for that request. The executive modifies the global variable SICUFT to point to this UFT. The executive clears any delay the SIC handler may have and sets the task status to ready so that it will run when its turn in the round robin schedule comes up.

When the SIC handler starts execution it calls the subroutine FUNEXE to obtain and decode the function requested out of the UFT and then call the proper function subroutine to execute the requested function. Each function has its own function subroutine. These subroutines call lower level subroutines, each of which perform a special service needed in processing the requested function. Table 9 gives an hierarchical presentation of the subroutines used in the SIC handler. Once the function request has been processed the status in the UFT (first byte of UFT) is changed to inform the calling task the status of its request. The subroutine FUNEXE then returns control to the main routine of this handler.

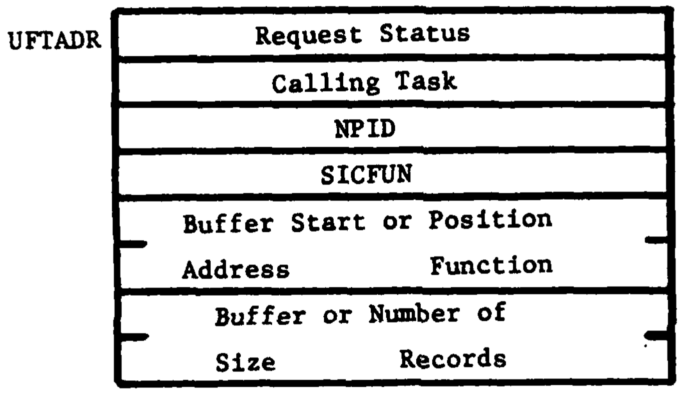


Figure 24 Format of SIC Handler's UFT

Table 8

SIC HANDLER REQUEST STATUS CODES

STATUS	CODE	DESCRIPTION
RQREJ	+2	Request rejected by executive because this handler is being used by another task.
BUSY	+1	The handler is in the process of executing the requested function.
SUCC	Ø	Requested function completed successfully.
INVFUN	-1	Invalid function requested.
COMERR	-2	SIC communication with nameplate in error.
FUNERR	-4	Nameplate error in execution of function.
INTERR	-5	SIC not properly initialized.
INVID	-1Ø	Invalid nameplate ID specified.
OVERFL	-11	The size of the data requested to be loaded is larger than the buffer area specified to receive it.
EDWM	-12	This status is set when trying to read, write or move the record pointer beyond the end of the nameplate's read/write memory.
BOWM	-13	This status is set when trying to move the record pointer before the start of the nameplate's read/write memory.

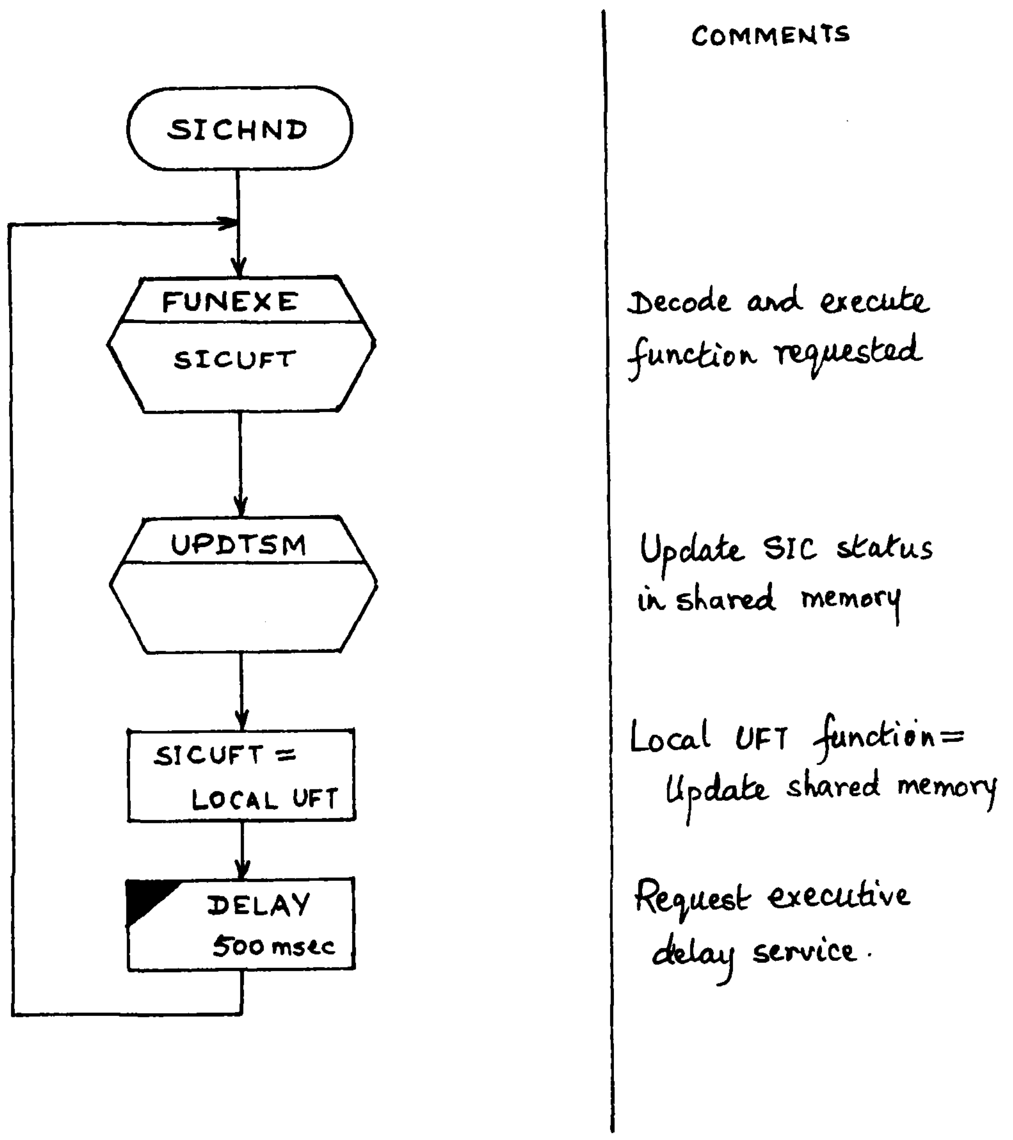

COMMENTS
SICHND
FUNEXE
SICUFT
Decode and execute function requested
UPDTSM
Update SIC status in shared memory
SICUFT = LOCAL UFT
Local UFT function = Update shared memory
DELAY 500 msec
Request executive delay service.

Figure 25 SIC Handler Task

Table 9

SIC HANDLER HIERARCHY

Main Routines (called by LM executive)	1st Level Subroutines	Function Subroutines	3rd Level Subroutines	4th Level Subroutines	5th Level Subroutines	6th Level Subroutines	7th Level Subroutines
HNDINT	UPDTSM	SICINT	BLDSIC	SELZRO	COMMUN	SEND	CKSUM
		LDDIR	DATALD	SELNXT	ERRCHK	RECEIV	SICTX
SICHND	FUNEXE	LDCFN	DATAWT	SELADR	BUSYCK		SICRX*
		LDCNV	DATAER	DESEL	ABX		
		LDDIAG		ASGADR			
		LDCAL		RDADDR			
		DMPWRT		RDMEM			
		RDREC		WRTEN			
		WRTREC		WRTMEM			
		ERAWRT		NXTADR			
		RECPOS		ERAMEM			
				RNDIAG			
				RDDIAG			
				NPABT			
				TABLMV			

*SICRX is really a "receiver data available" interrupt handler.

The subroutine UPDTSM is then called to check and update the SIC status byte in shared memory. Then the UFT pointer, SICUFT, is modified to point to a local UFT containing the function "update status in shared memory". The SIC handler then delays 500 milliseconds. If no task has requested an SIC handler function by the time this delay expires, the executive will activate this handler. The SIC handler will then proceed to execute the function in this local UFT. The subroutine FUNEXE merely returns control if it decodes the function "update status in shared memory". Then UPDTSM is called to perform this function. This scheme provides for a periodic update of the status in shared memory even if a handler function is not requested by a task.

Detailed information on all of the subroutines comprising the SIC handler is given in Appendix C, Section 2-A.

2.3.6 SHARED MEMORY HANDLER

The Shared Memory Handler (SMHND) runs as a task in the LM under control of the real-time executive. The three major functions of this handler are: (1) arbitrate SM buffer control for data transfers, (2) clear and set various flags, and (3) update the LM status byte in the SM. The handler runs at least every 0.5 seconds, since at the end of every run it issues an executive DELAY request of 0.5 seconds. This insures that even if no task activates the shared memory handler, it is periodically activated by the executive to update the LM status byte in SM. This LM status byte is not updated with every SMHND call but only after a minimum of 0.5 seconds.

The shared memory handler is activated through an executive request by another task. The reason for this is to relieve the calling task of the responsibility of needing to know any SM addresses or handshake protocols. This is particularly important for nonresident tasks which have no knowledge of any LM addresses or handshakes.

2.3.6.1 SM Handler Functions

Following is a description of each of the ten SMHND functions listed in Table 10. The various status returns from these functions are listed in Table 11.

UPDSTS (Update status) is function number Ø. This function is used to update the LM status byte in SM.

SETSTS (Set status) is function number 1. This function is used to write the calling task status byte into the LA status byte in SM.

SETFLG (Set flag) is function number 2. This function is used to set the flag bit in DXSTS byte in SM to indicate a calling task error condition.

CLRFLG (Clear flag) is function number 3. This function is used to clear the flag bit in DXSTS byte in SM.

S1READ (Read S1 (DXSTS)) is function number 4. This function is used by the calling task to read the DXSTS byte in SM.

SEQGET (Sequential get) is function number 5. This function is used to get sequential data from SM data buffer which has been loaded by the LMG.

REFGET (Refreshed get) is function number 6. This function is used to get refreshed data from SM data buffer which has been loaded by the LMG.

SEQPUT (Sequential put) is function number 7. This function is used to put sequential data into SM data buffer for the LMG to read.

REFPUT (Refreshed put) is function number 8. This function is used to put refreshed data into SM data buffer for the LMG to read.

Table 10

SMHND FUNCTIONS

Ø	UPDSTS	-	Update LM status.
1	SETSTS	-	Set task status.
2	SETFLG	-	Set flag bit.
3	CLRFLG	-	Clear flag bit.
4	S1READ	-	S1 byte read.
5	SEQGET	-	Sequential data get.
6	REFGET	-	Refreshed data get.
7	SEQPUT	-	Sequential data put.
8	REFPUT	-	Refreshed data put.
9	FLGINZ	-	Flags initialization.

Table 11

SMHND STATUS RETURN CODES

+1	Data not ready, or pending.
∅	Success.
-1	Invalid function number.
-2	BUFSIZ $<$∅ or $>$63.
-3	BUFSIZ too small (eg $<$ WSC).
-1∅	SEQPUT: RDY or REQ $\neq$ ∅

FLGINZ (Flags initialization) is function number 9. This function is used to initialize all the data transfer handshake flags.

2.3.6.2 SM Handler Calling Sequence

The shared memory handler is called by another task through an executive request. The executive builds a UFT table as shown in Figure 27 consisting of the parameters specified by the calling task, and passes it to the SMHND. The executive sets the SMHND task to the ready state and returns control immediately to the calling task. Thus the calling task must relinquish CPU control to give the SMHND a chance to run. The calling sequence is as follows:

```
    label EXECRQ SMRQ,SMFUN,BUFADR,BUFSIZ,UFTADR    where:
```

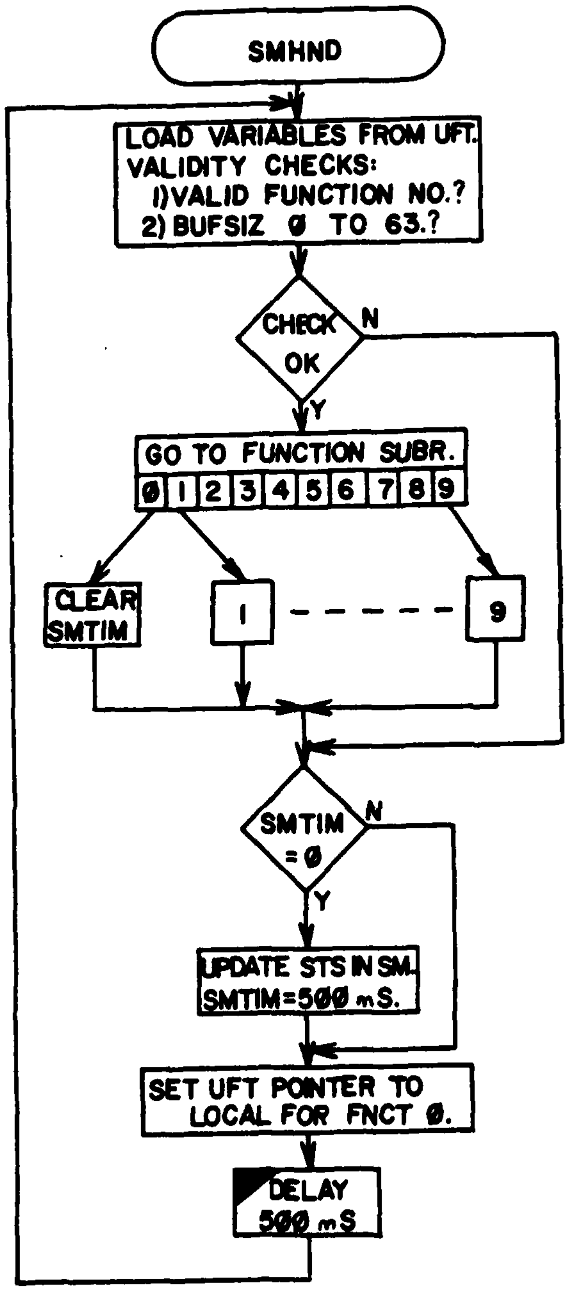
EXECRQ	- invokes the executive request macro.
SMRQ	- identifies EXECRQ as being for SMHND.
SMFUN	- identifies which of the ten functions.
BUFADR	- starting address of the user buffer.
BUFSIZ	- # of bytes available in user buffer.
UFTADR	- user file table containing above information to be passed to handler. See Figure 27.

Table 11 lists the request status return codes. The calling task may obtain the present status of the request by examining the first byte of the UFT and relinquish CPU control if the request has not completed.

2.3.6.3 SM Handler Design

The general handler architecture is shown in Figure 26. Upon being called it first checks the validity of the calling parameters. If valid then a call is made to the particular function subroutine which exe-

COMMENTS



BUFSIZ Ø TREATED AS 64.

UPDATE TASK DECREMENTS SMTIM.

UPDATE STS EVERY 5ØØmS, NOT EVERY FUNCTION CALL.

A TASK CALLING THE HANDLER WILL KILL THE DELAY.

Figure 26 SMHND Flowchart

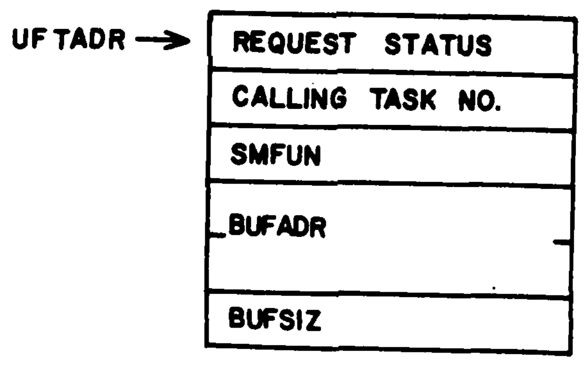


Figure 27 SMHND Calling Parameters Table

cutes the requested function. Then a check is made to see if the handler's local time of 0.5 sec has expired. If so, then the LM status byte in SM is updated and the timer is reset to 0.5 sec. Next the UFT pointer is set to point to the local UFT table containing the function request UPDSTS (#Ø) and an EXEC DELAY of 0.5 sec is requested. This insures that LM status is updated periodically between 0.5 and 1 sec. Refer to Appendix C, Section 3-F fo detail on the structure and workings for each function.

2.3.6.4 SM Communication Protocol

Figure 28 shows the organization and usage of the Shared Memory. There are 256 bytes divided into four 64 byte areas. Ø-63 is labeled IOBUFØ and is for data output from LMG to subsystem. 64-127 is labeled IOBUF1 and is for data input to LMG from subsystem. 128-191 is labeled LMBUF and is used for LM Function Command parameters between the LM and the LMG. 192-255 is used for all the control and status information between the LM and the LMG.

2.3.6.4.1 Function Command Protocol

The LMG-LM function command handshakes are handled by an interrupt routine rather than the Shared Memory Handler. Two bytes in Shared Memory are used for the function command handshakes: 1) Address FF hex is written by the LMG with bit 7 set to 1 and the command number in bit Ø - bit 6. 2) Address FD hex is written by the LM with the resulting command status code.

Bit 7 of address FF hex is used as a semaphore and set to 1 by the LMG when a command is written and is cleared to Ø by the LM to acknowledge receipt of the command. The sequence of events in a command handshake is as follows:

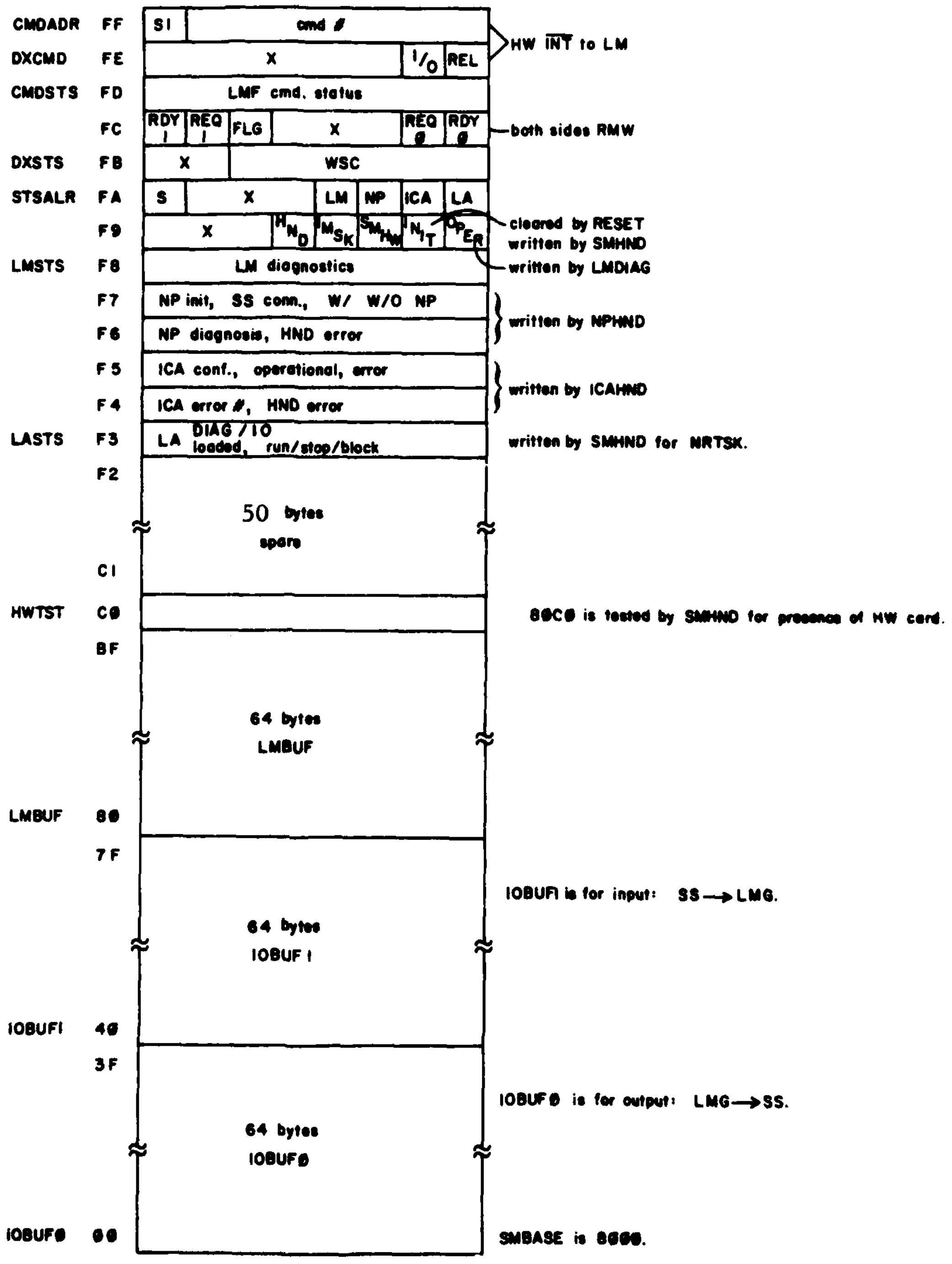

CMDADR FF SI cmd #
DXCMD FE X I/O REL
HW INT to LM
CMDSTS FD LMF cmd. status
FC RDY I REQ I FLG X REQ Ø RDY Ø
both sides RMW
DXSTS FB X WSC
STSALR FA S X LM NP ICA LA
F9 X HND MSK SMHW INIT OPER
cleared by RESET
written by SMHND
LMSTS F8 LM diagnostics
written by LMDIAG
F7 NP init, SS conn., W/ W/O NP
F6 NP diagnosis, HND error
written by NPHND
F5 ICA conf., operational, error
F4 ICA error #, HND error
written by ICAHND
LASTS F3 LA DIAG/IO loaded, run/stop/block
written by SMHND for NRTSK.
F2
50 bytes spare
C1
HWTST CØ
8ØCØ is tested by SMHND for presence of HW card.
BF
64 bytes LMBUF
LMBUF 8Ø
7F
IOBUFI is for input: SS → LMG.
64 bytes IOBUF I
IOBUFI 4Ø
3F
IOBUFØ is for output: LMG → SS.
64 bytes IOBUFØ
IOBUFØ ØØ
SMBASE is 8ØØØ.

Figure 28 SM Memory Map

1) LMG writes into address FF hex with bit 7 set to 1 and the command number in bits Ø through 6.

2) This generates an interrupt in the LM. The interrupt routine clears bit 7, checks the validity of the command number, and writes an intermediate status code 'command received' (+1) into address FD hex.

3) When the Command Interpreter executes, it changes the intermediate status to 'command active' (+2) in address FD hex.

4) When the Command Interpreter completes execution of the command it writes the final status code (Ø or-n) into address FD hex.

2.3.6.4.2 Data Transfer Protocol

Data transfers take place between the LMG and a subsystem. The software performing this in the LM is a nonresident task loaded from either the LMG or the NP. The nonresident task utilizes the SMHND to make data transfers to or from SM. Three bytes in SM are used in the data transfer handshakes: 1) Address FE hex written by LMG to cause LM to manipulate handshake bits, 2) Address FC hex contains the buffer control bits, and 3) Address FB hex contains the byte count for data transfers. There are four types of data transfers: Sequential input (SEQIN), Refreshed input (REFIN), Sequential output (SEQOUT), Refreshed output (REFOUT). Following is the handshake sequence for each of these.

SEQIN: 1) Nonresident task calls the SMHND function SEQPUT to put a buffer of data into SM.

2) SMHND transfers data and byte count from task buffer into SM.

3) SMHND sets RDY1 (b7 in FC) = 1.

4) SMHND sets asynchronous service request (b7 in FA).

5) SMHND returns 'pending' (+1) status to nonresident task.

6) LMG clears asynchronous service request (b7 in FA).

7) LMG requests the buffer (if RDY1=1, then RDY1=Ø and REQ1=1).

8) LMG reads the byte count and data from SM.

9) LMG issues STATUS function command to LM.

10) LM CMDITR function STATUS clears REQ1 to Ø.

11) Nonresident task calls the SMHND function S1READ to check REQ1 bit. Transfer is complete when REQ1 has been cleared to Ø.

REFIN:

1) Nonresident task calls SMHND function REFPUT to put a buffer of data into SM. If REQ1=1 then 'pending' (+1) status is returned to nonresident task and task relinquishes CPU control to try again soon.

2) SMHND clears RDY1 to Ø.

3) SMHND transfers data and byte count from task buffer into SM.

4) SMHND sets RDY=1.

5) SMHND returns 'success' (Ø) status to nonresident task.

6) LMG requests buffer (if RDY1=1 then RDY1=Ø and REQ1=1).

7) LMG reads byte count and data from SM.

8) LMG writes DXCMD (address FE hex) = Ø3 causing an interrupt to the LM.

9) LM interrupt routine sets REQ1=Ø and RDY1=1.

SEQOUT: 1) LMG requests buffer (if RDYØ=1 then RDYØ=Ø and REQØ=1).

2) LMG writes byte count and data into SM.

3) LMG writes DXCMD (address FE hex) = ØØ causing interrupt to the LM.

4) LM interrupt routine sets REQØ=Ø and internal 'DATRDY' flag=1.

5) Nonresident task calls SMHND function SEQGET to get data. If DATRDY = Ø then 'pending' (+1) status is returned to the nonresident task and the task relinquishes CPU control to try again soon.

6) SMHND transfers data and byte count to task buffer.

7) SMHND clears internal DATRDY flag to Ø.

8) SMHND sets RDYØ=1.

9) SMHND returns 'success' (Ø) status to nonresident task.

REFOUT: 1) LMG requests buffer (if RDYØ=1 then RDYØ=Ø and REQØ=1).

2) LMG writes byte count and data into SM.

3) LMG writes DXCMD (address FE hex) = Ø1 causing interrupt to the LM.

4) LM interrupt routine sets RDYØ=1 and REQØ=Ø.

5) Nonresident task calls SMHND function REFGET to get data. If REQØ=1 then 'pending' (+1) status is returned to the nonresident task which relinquishes CPU control to try again soon.

6) SMHND sets RDYØ=Ø.

7) SMHND transfers data and byte count to task buffer.

8) SMHND sets RDYØ=1.

9) SMHND returns 'success' (Ø) status to nonresident task.

2.3.7 INTERRUPT SERVICE ROUTINE

The LM Motorola 6800 processor has three different types of interrupts: Reset, Nonmaskable Interrupt (NMI), and Interrupt Request (IRQ). Reset is used during power up or as a result of the reset pushbutton being depressed. This action causes the hardware to be initialized and the software to execute from an initialization program. NMI is not used. IRQ is the only remaining interrupt source and is used to service all system interrupts. Interrupts are enabled by clearing the interrupt mask bit in the condition code register (6800 instruction CLI). Execution of the interrupt service routine (INT) takes place whenever interrupts are enabled and IRQ pin goes low.

The sources for an IRQ are listed in Table 12. There is a subroutine to service each of these sources and INT makes a call to each of these as shown in Figure 29. Since each subroutine is called for every interrupt, each subroutine must check for its interrupt conditions and simply return immediately if its execution is not required.

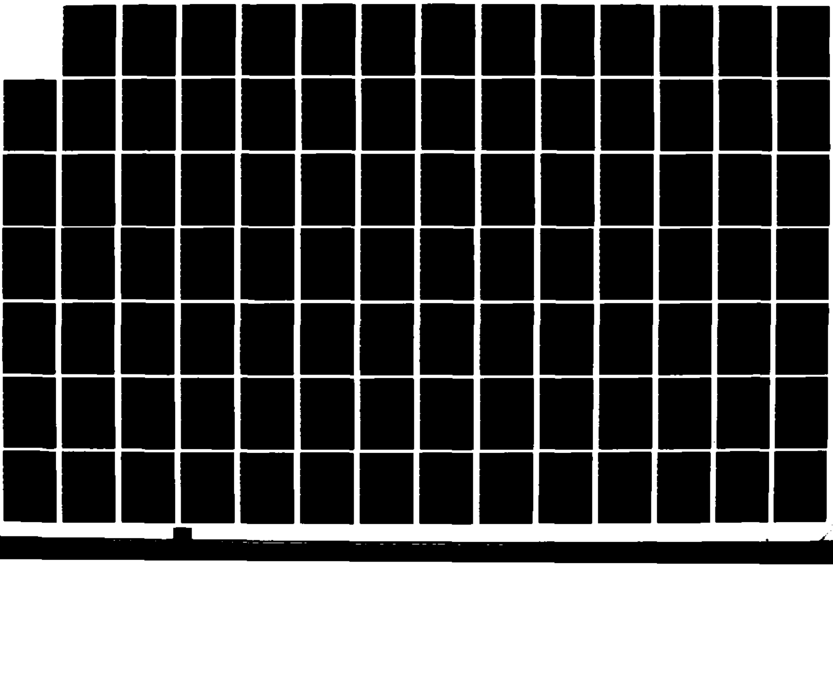
All machine registers are automatically saved by the 6800 hardware upon an interrupt and are automatically restored upon execution of a Return from Interrupt (RTI) instruction. A detailed description of the interrupt routines is presented in Appendix C, Section 2-G.

2.3.8 COMMAND INTERPRETER

The Command Interpreter (CMDITR) task runs in the LM and interprets twelve (12) distinct commands from the LMG. These commands are listed with a short description in Table 13. The LMG issues a command by writing the corresponding code into the command byte (address FF) in shared memory. Status of the command is returned to the LMG in the command status byte

AD-A124 621 THE REMOTE LINK UNIT: A DEMONSTRATION OF OPERATIONAL PERFORMANCE PART III..(U) HOUSTON UNIV TX DEPT OF ELECTRICAL ENGINEERING C J TAVORA ET AL. AUG 81 2/3

UNCLASSIFIED AFWAL-TR-81-1131-PT-3-1 F33615-80-C-1095 F/G 17/2 NL



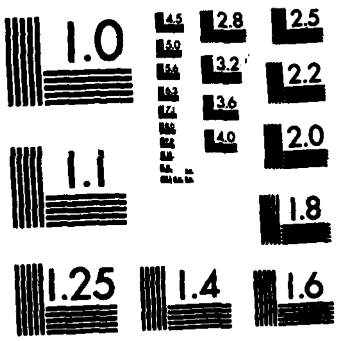

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

Table 12

IRQ SOURCES

1.	Time
2.	CMDINT
3.	DXINT
4.	INTPIA
5.	NP RX

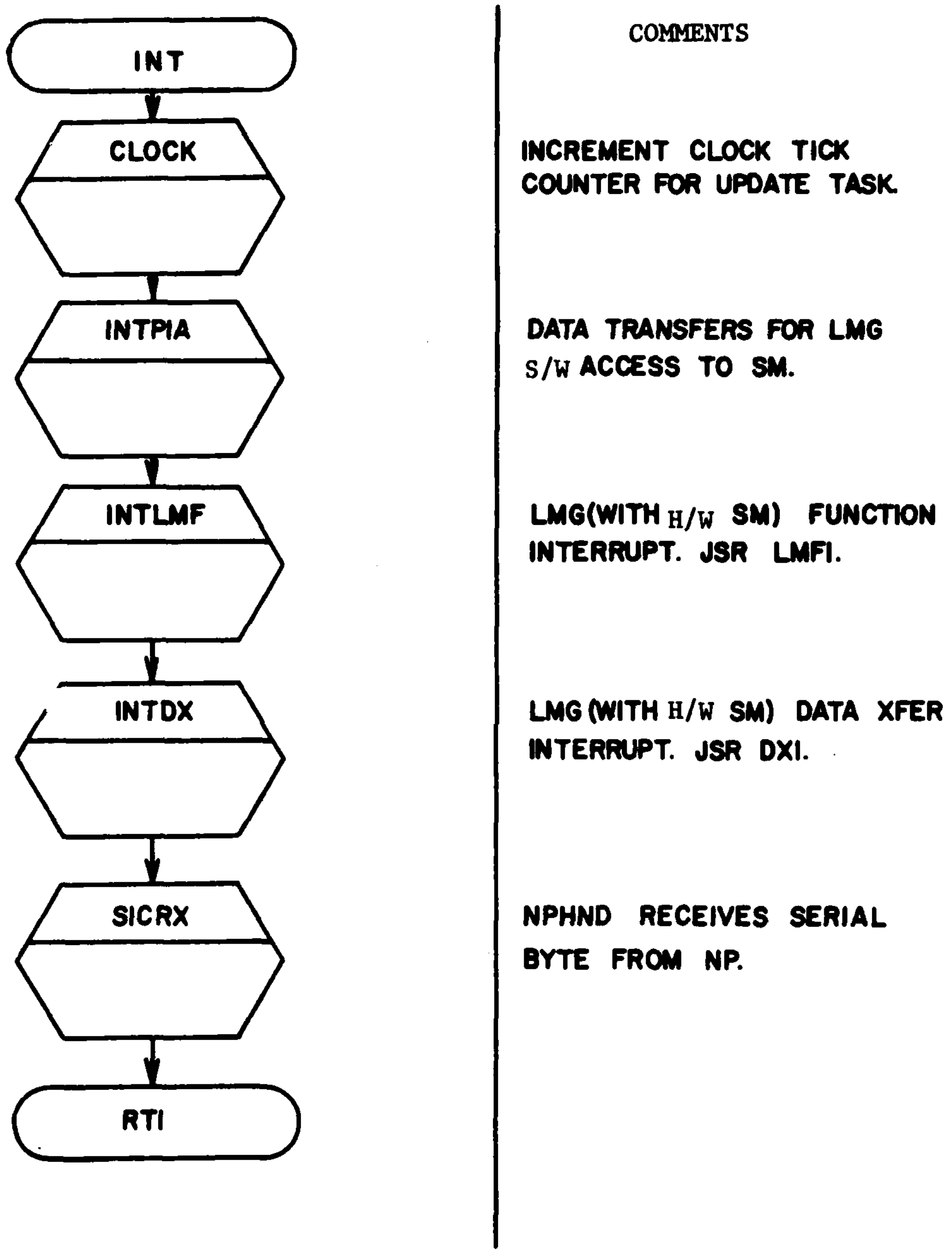

COMMENTS
INT
CLOCK
INTPIA
INTLMF
INTDX
SICRX
RTI
INCREMENT CLOCK TICK COUNTER FOR UPDATE TASK.
DATA TRANSFERS FOR LMG S/W ACCESS TO SM.
LMG(WITH H/W SM) FUNCTION INTERRUPT. JSR LMFI.
LMG (WITH H/W SM) DATA XFER INTERRUPT. JSR DXI.
NPHND RECEIVES SERIAL BYTE FROM NP.

Figure 29 INT Flowchart

Table 13

LM COMMANDS

PRGMLD	81	Load non-resident task.
RUN	82	Run non-resident task.
STOP	83	Stop non-resident task.
NPDIAG	84	Same as NPINIT.
	85	Reserved.
CANCEL	86	Cancel the previous cmd. that is pending.
XFRTBL	87	Transfer LM system tables to/from LMG.
STATUS	88	Clear input buffer request bit.
RESET	89	Resets CMDITR and ICA.
RESTRT	8A	Jump to power up restart location.
NPINIT	8B	Request NPHND to initialize NP's.
CONFIG	8C	Configure selected subsystem.
NOOP	8D	No operation.

(address FD) in shared memory and are listed in Table 14. The commands are mutually exclusive and a new command may not be issued until the previous command has been completed, with the exception of CANCEL and RESTRT, which may be issued anytime.

The command interpreter task shown in Figure 30 works in conjunction with the LM function interrupt routine shown in Figure 31. When the LMG writes into the command byte in shared memory, the resulting interrupt initiates the LM function interrupt routine. The routine checks the command for validity or express (CANCEL, RESTRT) and then does one of four things, depending upon the command and the state of the command interpreter task: 1) if the command is express (ie CANCEL or RESTRT), then it is executed to completion; 2) if a previous command is in progress, then an error status (-2) is returned to the LMG; 3) if the command is invalid, then an error status (-1) is returned to the LMG; 4) otherwise, a flag is set up for the command interpreter task to recognize and begin execution. A 'command received' status is returned to the LMG.

The command interpreter task in its quiescent mode continually checks for the new command flag set by the LM function interrupt routine and relinquishes CPU control if the flag is not set. When the flag is found set, the command status to the LMG is changed from 'command received' to 'command active'. The interpreter task then calls one of twelve command modules which return with a module status. This status is copied to the command status in shared memory and the new command flag is cleared. The interpreter task then returns to its quiescent mode waiting for another command.

A description of each of the twelve commands will now be presented.

Table 14

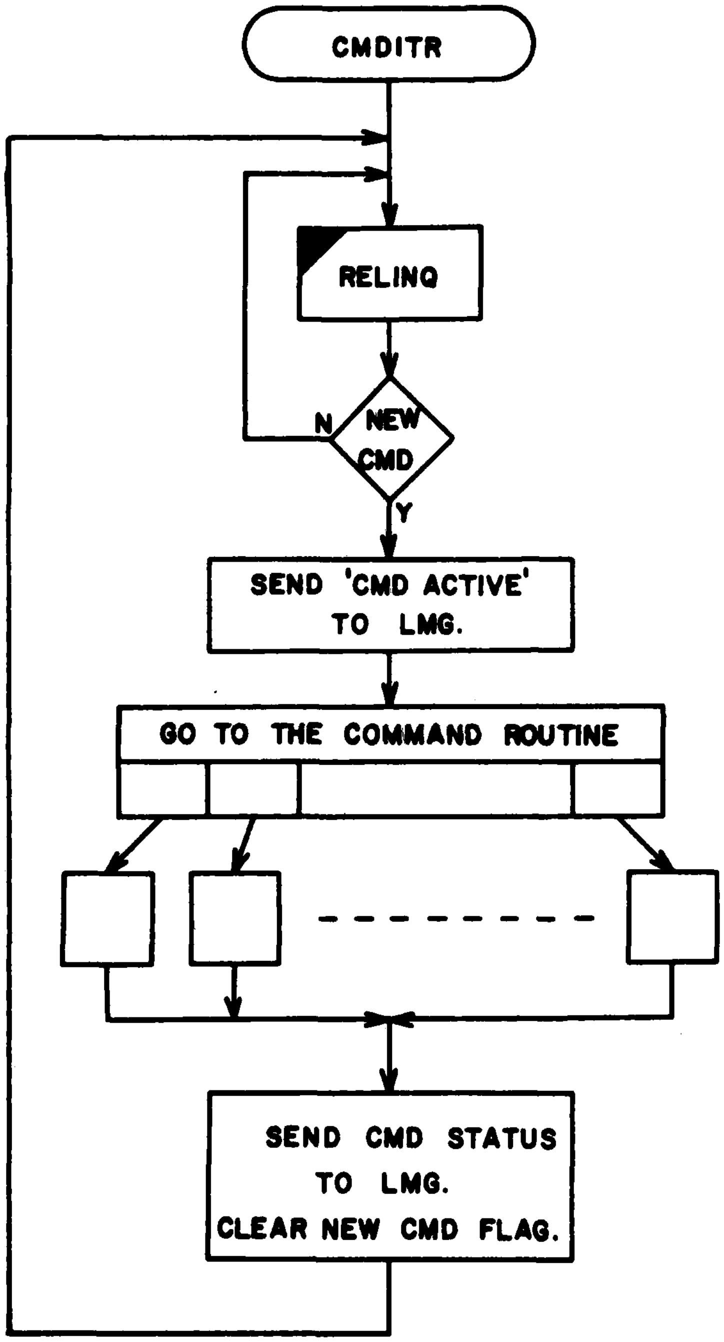
CMDITR STATUS RETURN CODES

Command	Status	Indication
General	+2	Command active.
	+1	Command received.
	Ø	Success.
	-1	Invalid command.
	-2	Another command still in progress.
	-3	Command not implemented yet.
	-4	Command cancelled.
PRGMLD	-1Ø	# of bytes not from -1 to +61.
	-11	Existing non-resident task is not dormant.
	-12	TYPE not Ø or 1.
	-13	SOURCE not Ø or 1.
	-15	XFR or TRLR record with SOURCE = NP.
	-16	Invalid start address.
	-17	Invalid end address.
	-18	XFR record with no HDR record.
	-19	TRLR record with no XFR record.
	-2Ø	EXEC failure to REMOVE previous task.
	-21	NP failure to upload task.
	-22	EXEC failure to INSTAL NP task.
	-23	Program checksum not zero.
	-24	EXEC failure to INSTAL LMG task.
RUN	-1Ø	A task is not installed.
	-11	The task is not dormant.
	-12	Task checksum not zero.
STOP	-1Ø	A task is not installed.
	-11	The task is dormant.
	-12	Failure to STOP after 5 sec.
NPDIAG	-	see NPINIT.
CANCEL	-	None
XFRTBL	-1Ø	Invalid table number.
	-11	To/from LM not Ø or 1.
	-12	# bytes not from 1 to 64.
	-13	Offset <Ø.
	-14	Attempt to write a read only table.
	-15	TIME: # bytes too large.
	-16	TIME: Offset too large.
	-17	NPREC: Function not Ø to 5.
	-18	NPREC: Invalid # bytes.
	-19	SICSTS: Table shows <1 # of NP's.
	-2Ø	SICSTS: Table shows >2Ø # of NP's.

Table 14 (cont.)

CMDITR STATUS RETURN CODES

Command	Status	Indication
XFRTBL	-21	NPHND error.
	-22	ICCNFG: ICAHND error GRP A.
	-23	ICCNFG: ICAHND error GRP B.
STATUS	-	None
RESET	-1Ø	ICAHND fail to reset ICA.
RESTRT	-	None
NPINIT	-1Ø	NPHND error.
CONFIG	-1Ø	SOURCE not Ø or 1 (LMG or NP).
	-11	GROUP not 1 or 2 (A or B).
	-12	ICAHND error.
	-13	NPHND error.
	-14	Invalid NP table.
NOOP	-	None


CMDITR
RELINQ
NEW CMD
N
Y
SEND 'CMD ACTIVE' TO LMG.
GO TO THE COMMAND ROUTINE
SEND CMD STATUS TO LMG. CLEAR NEW CMD FLAG.

COMMENTS

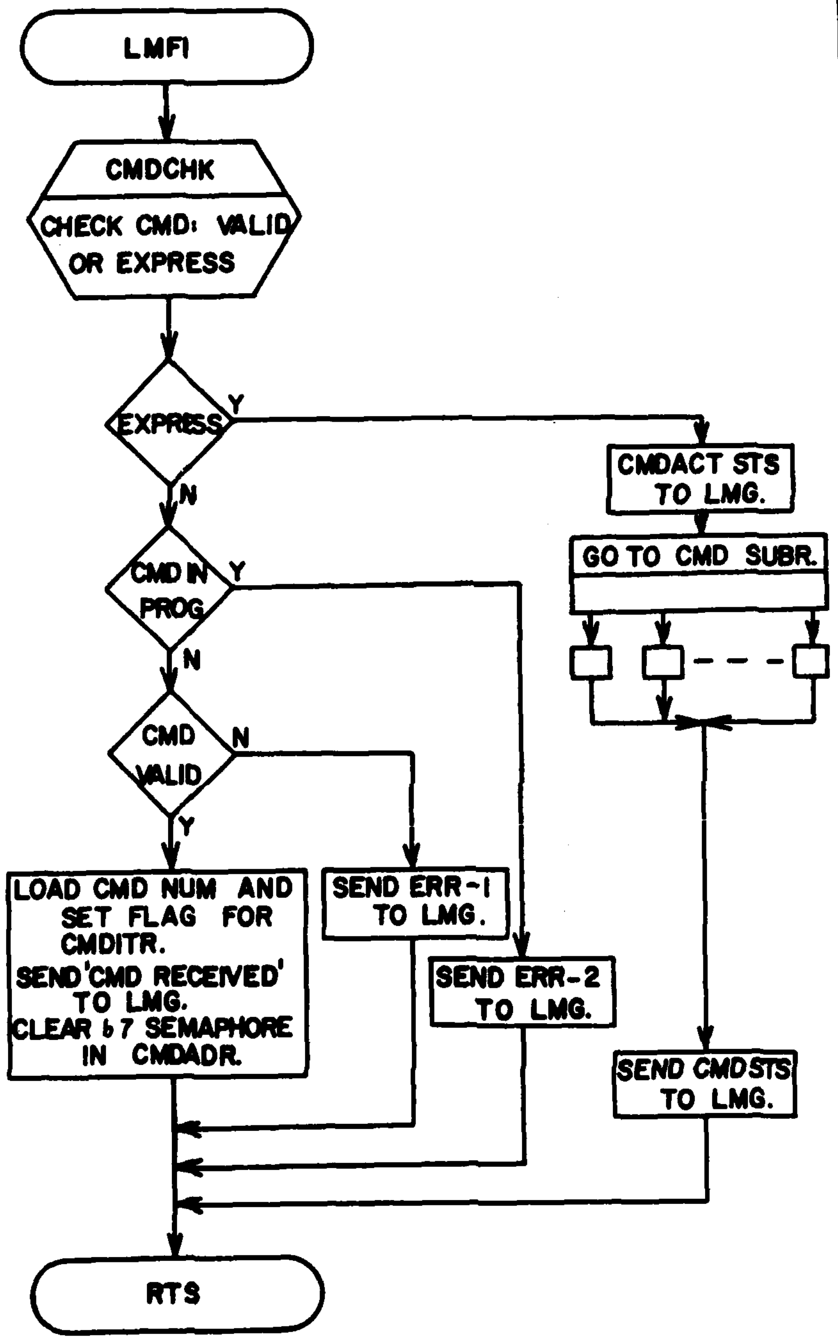
WAIT FOR NEW COMMAND FLAG WHICH IS SET BY LMFI WHEN A NEW CMD IS RECEIVED FROM THE LMG.

13 COMMAND TABLE ENTRIES. ONLY THOSE LABELED 'NOEXPR' ARE DONE HERE.

EACH ROUTINE RETURNS WITH CMD STATUS.

Figure 30 CMDITR Flowchart

COMMENTS



13 CMD TABLE ENTRIES. ONLY ENTRIES LABELED EXPR ARE DONE HERE. EACH ROUTINE RETURNS WITH A CMD STATUS.

Figure 31 LMFI Flowchart

PRGMLD (Program Load) is command number 81 hex. This command is used to load programs into the nonresident task area in the LM from either the NP or the LMG and then to install the loaded program as a task with the executive.

RUN (Run nonresident task) is command number 82 hex. This command is used to activate the nonresident task for running as part of the real-time system.

STOP (Stop nonresident task) is command number 83 hex. This command is used to stop the execution of the LM nonresident task.

NPDIAG (NP diagnostic) is command number 84 hex. This command is used to cause the Nameplates to run their internal diagnostics. NPDIAG is identical to NPINIT and is implemented as a jump to NPINIT.

Command number 85 hex is reserved.

CANCEL (Cancel) is command number 86 hex. This command is used to stop the execution of any other command module already in progress.

XFRTBL (Transfer table) is command number 87 hex. This command is used to transfer all or part of any one of eleven LM tables to or from the LMG.

STATUS (Status) is command number 88 hex. This command is used to assist in the data transfer handshake for sequential input to the LMG by clearing the REQ1 bit in the S1 byte in SM (address FC hex).

RESET (Reset) is command number 89 hex. This command is used to reset the state of the CMDITR task and the ICA hardware.

RESTRT (Restart) is command number 8A hex. This command is used to cause the LM to jump to its power up restart location.

NPINIT (NP initialization) is command number 8B hex. This command is used to cause the NPHND to reset all the NP's, assign addresses to the NP's, and cause each NP to run its internal diagnostic.

CONFIG (ICA configure) is command number 8C hex. This command when used causes the ICAHND to configure either group A or group B of the ICA with configuration parameters from either the LMG or the NP.

NOOP (No operation) is command number 8D hex. This command is used to provide the LMG with means of exercising the command handshake without causing anything to happen.

The parameters associated with each command are listed in Table 15. Interactions with the rest of the system are shown in Figure 32. Refer to the User's Manual for more detail on usage of the commands. Refer to Appendix C, Section 2-H.

2.3.9 NONRESIDENT SOFTWARE

The LM has 2K bytes of read/write memory (addresses Ø4ØØ to ØBFF) allocated for loading external programs for execution. An external program may be either uploaded from a subsystem nameplate or downloaded from the LMG. Although there are many types of programs (data I/O, subsystem diagnostic, calibration), only one program may be loaded at any one time. This program is treated as an independent task.

A typical LMG command sequence to the LM might be as follows:

1) Issue command STOP to halt the present nonresident task. The STOP command module will set a STPRQ flag for the NRTSK to executive request EXIT.
2) Issue command PRGMLD, which will load an external program into the LM nonresident task area from either the NP or the LMG. The PRGMLD command module will perform the following

Table 15

LM COMMAND PARAMETERS

PRGMLD header record:	8Ø	FF	
	81	TYPE	ØØ - I/O program
			Ø1 - DIAG program
	82	SOURCE	ØØ - LMG
			Ø1 - NP
PRGMLD transfer record:	8Ø	# bytes, Ø1 to 3D	
	81	starting load address, high byte	
	82	starting load address, low byte	
	83 to BF	program bytes	
PRGMLD trailer record:	8Ø	ØØ	
	81	program checksum	
XFRTBL:	8Ø	table #, Ø1 to ØB	
	81	# bytes	
	82	offset into table	
	83	to/from	ØØ - LM to LMG
			Ø1 - LMG to LM
	84 to BF	table loaded by LM or LMG depending on to/from byte	
CONFIG:	8Ø	SOURCE	ØØ - LMG
			Ø1 - NP
	81	GROUP	Ø1 - GRPA
			Ø2 - GRPB
	82 to 89	configuration data if from LMG	

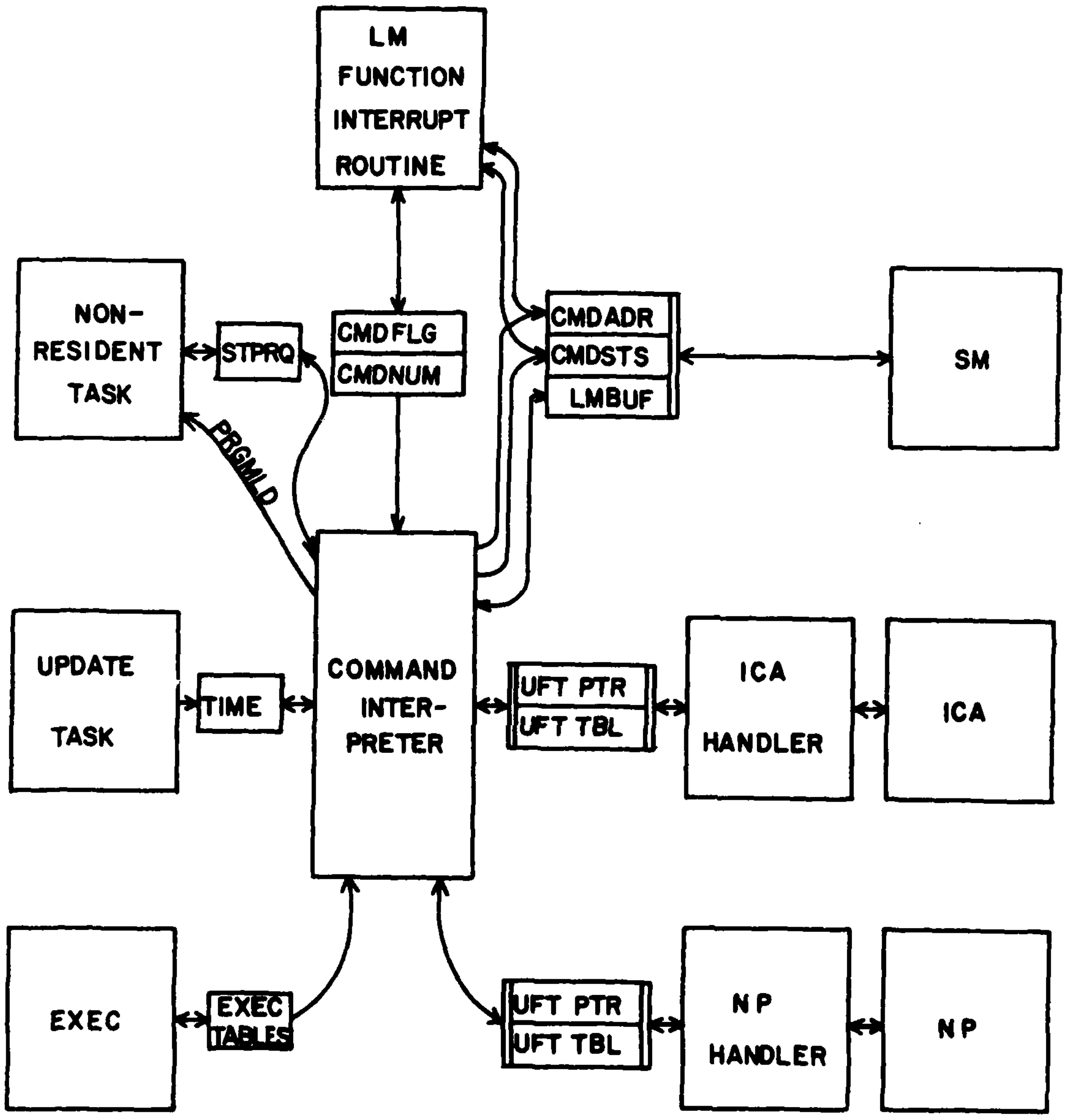


Figure 32 Command Interpreter Interactions

functions:

a) Request the executive to REMOVE the previous task from the active task list.

b) Load the new program into the nonresident task area.

c) Verify the new program's checksum.

d) Request the executive to INSTAL the new task on the active task list.

3) Issue command RUN to start the nonresident task. The RUN command module will issue the executive request ACTVAT.

The nonresident task may use resident LM resources. These include SM, ICA, and NP handlers, data and time of day, and in particular executive-supplied math functions. As an example, for this demonstration there is a synchro input/output task which when running requires the following services and resources:

EXECRQ ICARQ for synchro input voltages.

EXECRQ MATH for voltage checks and calculations.

EXECRQ ICARQ for synchro output voltages.

EXECRQ SMRQ to send degrees data to LMG.

EXECRQ DELAY for loop timing between outputs.

EXECRQ SICRQ to record errors.

TIME to record time of error.

In order to properly load and execute a nonresident task, the first thirteen bytes of the nonresident task must be a header of the form:

1) to 6) Program name - 6 ASCII characters

7) Start address H

8) Start address L

9) End address+1 H

10) End address+1 L

11) Initial stack pointer H

12) Initial stack pointer L

13) Reserved for insertion of program checksum

These parameters are used by the system during both loading and activation for execution.

SECTION 3

INTERFACE CONFIGURATION ADAPTER

The Interface Configuration Adapter (ICA) is the component that provides the LM with its universal interfacing capability. The RLU, working with Electronic Nameplates, can identify interfaced subsystems and automatically configure the ICA with the appropriate electrical interface.

The ICA can be used with a wide variety of I/O signal types. The interface consists of two groups of four I/O channels. Each group can be independently programmed to support either:

1) 4 AC or DC analog input or output lines, or
2) 4 parallel digital input or output lines, or
3) 1 serial synchronous digital input or output channel with handshaking, or
4) 1 synchro input or output channel, or
5) a variety of self-test terminations.

The ICA can be described in terms of the following 7 major hardware sections:

1) Signal input and output (SIO) channels for Group A.
2) SIO channels for Group B.
3) Reference Generation (shared by both groups).
4) Address decoding and configuration control (ADCC) for Group A.
5) ADCC for Group B.
6) Serial input and output (SERIO) circuitry for Group A.
7) SERIO circuitry for Group B.

These sections and their interconnections are indicated in Figure 33.

3.1 SIGNAL I/O CHANNEL DESIGN

A block diagram of a signal input/output (SIO) channel is shown in Figure 34. The channel provides the required functions of digital-to-analog conversion, analog-to-digital conversion, amplification, buffering, sampling and logic level processing. The SIO channels interface to the LM through several control lines, read/write strobe lines and the ICA bus.

The SIO topology is unique in that it provides a signal wrap around feature. Each SIO channel contains an ADC which is continuously updated at an 800 samples per second rate. This ADC can be read at any time by the LM to measure the input level if the SIO channel is programmed for input or to measure the output level if the SIO is programmed for output.

The input and output functions are completely independent in the SIO, sharing only the common subsystem bound I/O lines (+/-). With the SIO channel programmed for output the input can be used to perform several internal test functions or to monitor the output. The output can be programmed as single-ended (with the return through the SIO ground line) or as differential. This programming has no effect on the input configuration.

The output signal is derived from one of three sources:

1) the output DAC,

2) the group HILEVEL signal line, or

3) the group LOLEVEL signal line.

This selection is through the output MUX as shown in the Figure 34 . Addressing of the MUX is controlled by the ADCC section of the ICA. For

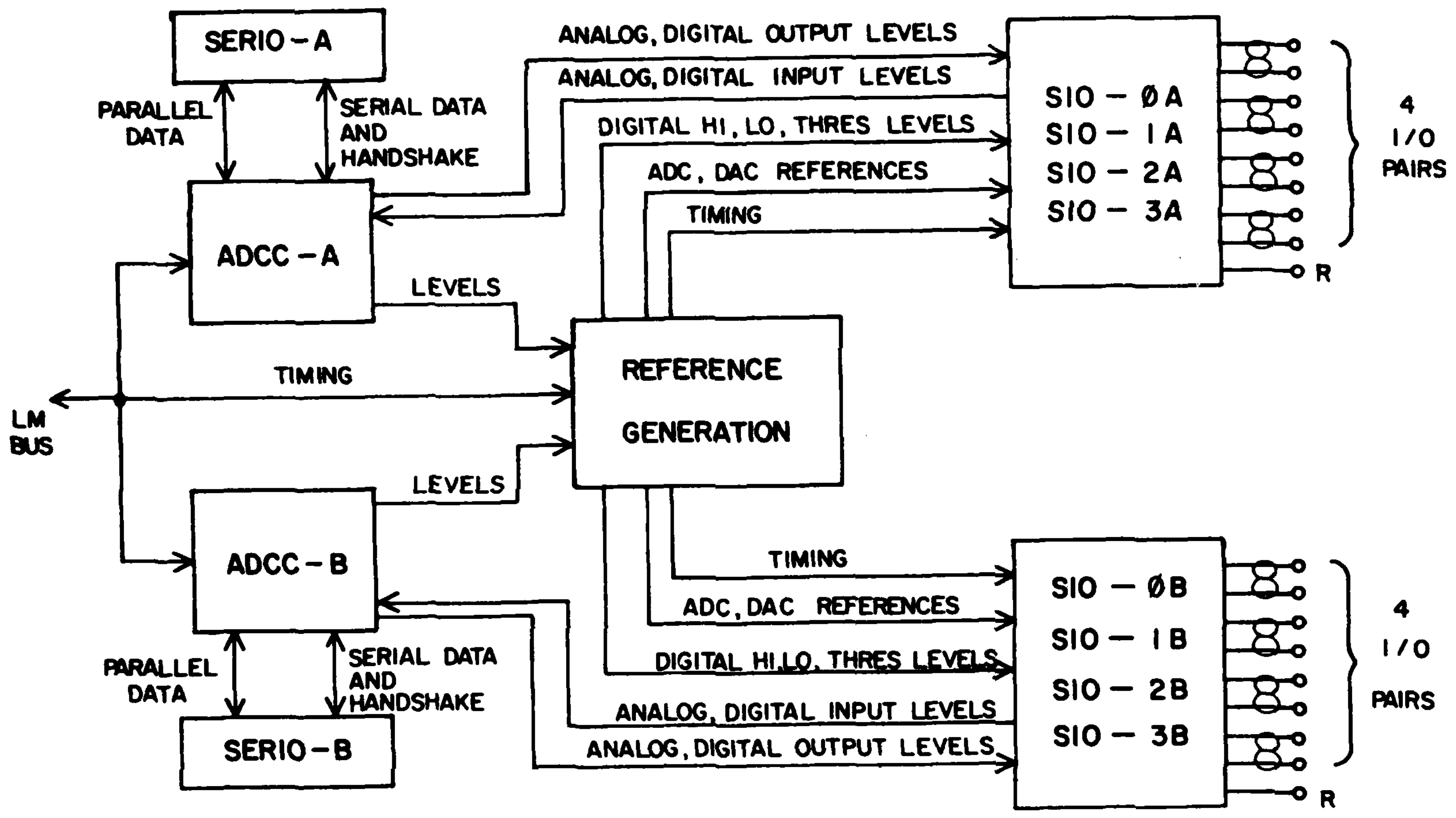


Figure 33 Decomposition of the ICA into Sections

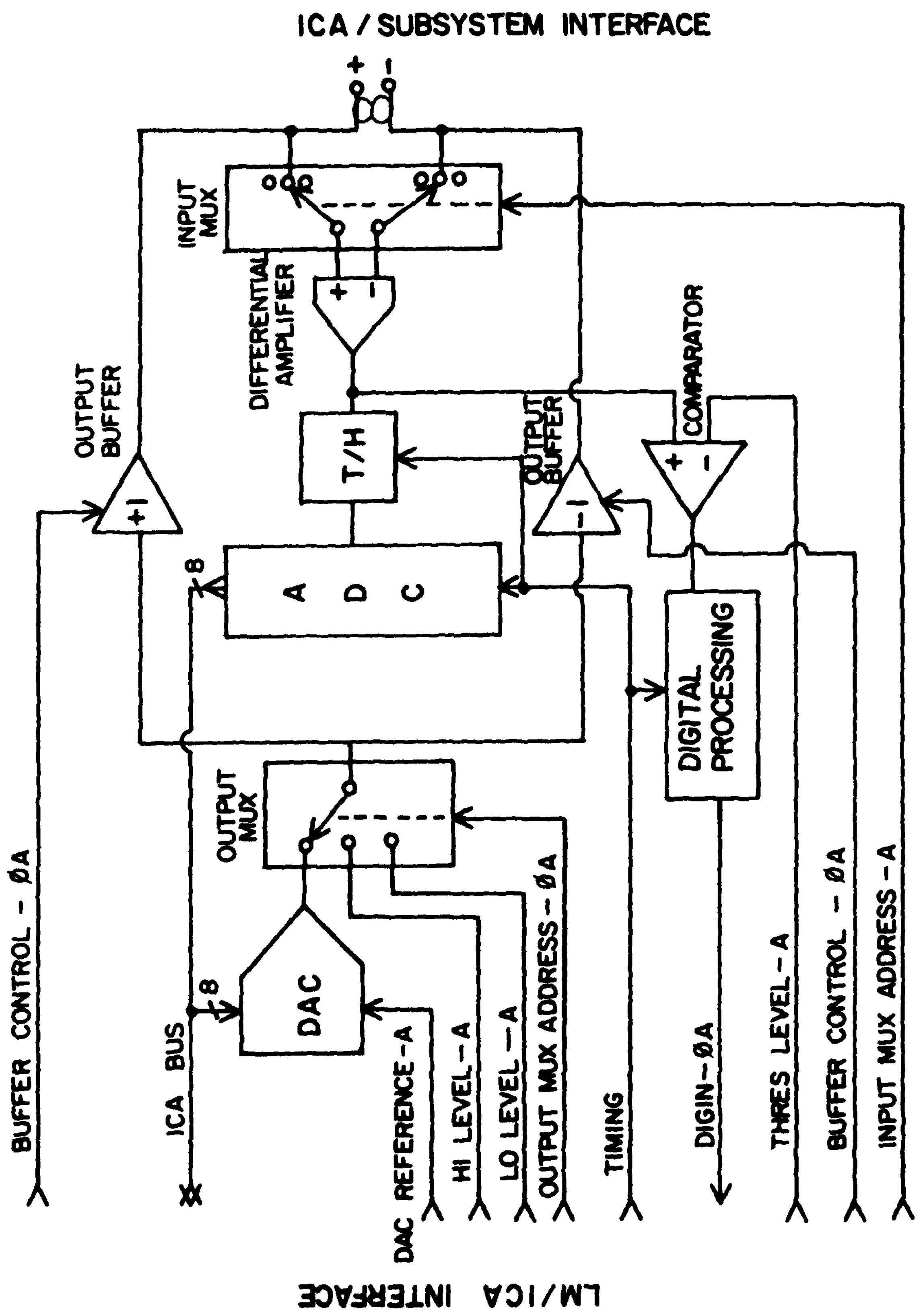


Figure 34 Block Diagram of a Signal I/O Channel

analog output, the output DAC is selected. The reference applied to the DAC originates in the ADCC section of the ICA. For DC outputs the DAC reference is 10 volts. For AC outputs the DAC reference is a 400 Hz sinewave of fixed amplitude. For digital output the MUX is addressed to the programmed HI or LO logic level depending upon the desired output logic signal.

Digital inputs are processed through the normal input MUX and Differential Amplifier. The logical threshold is programmed for the group as the THRESLEVEL shown in the Figure 34. Following level slicing, the logic input is processed and made available to the LM on the DIGIN line. This section is always operational so that simple level detection can be accomplished in either the digital or analog modes.

Appendix B, Section 3-A contains a detailed schematic of the Signal I/O channel and will be referred to in the discussion below. The discussion will focus on the operation of Channel Ø but will of course be applicable to each of the four channels in a group.

3.1.1 ANALOG INPUT PROCESSING

The analog input signal processing is accomplished by the input multiplexer M1, the input amplifier M2, the track and hold circuitry consisting of M15 and M8 and the channel analog to digital converter (ADC) M11.

The input multiplexer is programmed to either address 3 or 7 to allow input signals to reach the input amplifier. Note that the input amplifier is an instrumentation amplifier capable of implementing either a single-ended amplifier or a differential amplifier. Address 3 results in a single-ended input with the positive (+) input lead going to the positive input of the amplifier and the negative (-) input lead being open circuited.

The signal return is assumed to be provided via the group signal return line of the interface cable. Additionally address 3 uses the multiplexer to ground the negative input of the amplifier. Address 7 results in a differential input configuration with the + input lead going to the + input of the amplifier and the - input lead going to the - input of the amplifier.

The common mode limit of the input amplifier (+/- 10 volts) must be observed by any input signals. The amplifier is protected from overloads at the input by the protectior circuitry of R1, R2 and D1-D4. The amplifier provides a gain of one to any differential signal applied to its inputs. The common mode rejection ratio of the input amplifier is specified as greater than 70 dB at a gain of one. The input impedance of the input amplifier is greater than 3×10**9 ohms. The amplifier output is applied to the input of the track and hold circuit.

The track and hold circuit provides the signal processing required by the analog to digital converter. For DC input signals the track and hold circuit is used in a conventional manner. The +/- line from the ADCC section is held in the + state which results in the track and hold circuit having a gain of one when in the track mode. The signal is sampled and the sample converted by the ADC at an 800 samplesper second rate. The hold/ track modes are controlled by the H/T line from the ADCC. For AC signals the +/- line is used to invert the gain of the track and hold circuit at an 800 Hz rate. The gain is made positive during the positive half cycle of the ICA 400 Hz reference signal and is made negative during the negative half cycle. The H/T signal takes a sample at the positive and negative peak of each cycle of the 400 Hz reference. If the input AC signal is derived from the ICA 400 Hz reference the circuit operation results in sampling the peak value of the input waveform and obtaining a sample whose polarity is indicative of the phase of the input.

The ADC chip used is designed to accept inputs of 0 to + 5 volts and to provide a straight binary conversion on these inputs. The ICA is designed to process signals in the range of +/- 10 volts. The output of the track and hold circuit is scaled by resistors R13, R14, and R15 to accommodate the ADC input requirements. The ADC output count relates to the input voltage through the formula

$$V_{in} = -12.8 + 0.1*(\text{COUNT})$$

The correspondence between significant voltages and counts are given below.

Count		Voltage
Decimal	Hexadecimal	
0	∅	-12.8
28	1C	-10.0
128	8∅	0.0
228	E4	+10.0
255	FF	+12.7

The ADC runs continuously in any of the ICA modes. The ADC chip has an internal register which is updated at the 800 Hz sampling rate described above and which can be accessed at any time by the LM.

3.1.2 DIGITAL INPUT PROCESSING

Digital inputs are processed through the input amplifier M2 in the same manner as are analog inputs. This provides for a high input impedance for the digital input as well as the capability of either single-ended or differential processing as described above. The output of the input amplifier is connected to an analog comparator M7. The comparison threshold is programmed in the reference generation circuitry described in Section 3.2. The comparison threshold can be programmed between +/- 10 volts. This allows the ICA to process inputs from virtually any standard logic family. The resulting logic states at the output of M7 reflect whether the input is above (logic one) or below (logic zero) the programmed threshold value. Negative logic can be interpreted by proper handling in the LM via programs in the Electronic Nameplate.

Logic processing circuitry for the SIO channel consists of M9 and M10 which can be programmed to two distinct modes. In the follow mode the output of the comparator is continuously sampled at a 500 KHz rate by M9A. The Q output of M9A is multiplexed to the DIGIN-OR line by M10 and is readable by the LM by means of circuitry described in Section 3.3 below. The DIGIN-OR line simply represents the most recent sample of the comparator output and will follow this output continuously. In the latch mode the sampled comparator output is used in latch M9B. If the input crosses the programmed comparison threshold from above M9B will be set. The state of M9B is multiplexed to the DIGIN-OR line by M10 and is readable as before. Each time the four DIGIN lines of a group are read by the LM, circuitry in the ADCC section of the ICA resets the input latches (register M9B) for the entire group. This mode is designed to process momentary signals such as produced by contact closures.

3.1.3 CONTACT CLOSURE PROCESSING

Contact closures, either floating or to ground, can be detected in either of the two modes described in Section 3.1.2.

For floating contacts, a differential input is configured by the input MUX M1 and the internal Thevenin sources are placed on the I/O lines by activating analog switch M3. This results in a balanced +/- 5 volt source with a 6.7 K ohm source impedance which is applied across the input leads. If a contact is closed across the input pair the input voltage is reduced from 10 volts to 0 volts. A programmed threshold of 5 volts applied to M7 will result in the state of the contacts being represented by the logic level at the output of M7. This level can be processed as any other digital input.

Contacts to ground are processed by selecting a single-ended input configuration and setting the logic threshold to 2.5 volts. The external contact is placed from the + lead to ground and sees a Thevenin source in the ICA of 5 volts and 3.33 K ohms. The contact closure reduces the output of M2 from +5 volts to 0 volts.

3.1.4 ANALOG OUTPUT PROCESSING

The analog output signal processing is accomplished by the output digital-to-analog converter (DAC) components M13 and M14, the output multiplexer M12 and the output buffer amplifiers comprised of M4, M5 and M6. The analog output value is determined by two parameters which are input to the DAC: the output signal type (DC or AC) determined by the reference input to the DAC and the output signal amplitude determined by the digital input to the DAC. The DAC is a four quadrant multiplying type and as such can provide output signals of the form

$$V_{out} = Vref*(128-COUNT)/128$$

where Vref is the reference voltage applied to the DAC and COUNT is the value of the digital input to the DAC. COUNT is interpreted as varying between +127 and -128 (offset binary). The table below illustrates the correspondence between COUNT and special voltage values.

Count		Voltage
Decimal	Hexadecimal	
0	ØØ	10.0
1	Ø1	9.92
128	8Ø	0.0
255	FF	- 9.92

For single-ended DC output the reference voltage Vref is set to +10.00 volts. The available analog output ranges from -9.92 to +10.00 volts. The single-ended state is detected in the ADCC circuitry by logic which determines the number of output buffer amplifiers which are enabled (one => single-ended, 2 => differential outputs). When two channel buffers are enabled, the value of the voltage Vref is reduced to +5.00 volts. The differential output ranges from -9.92 to +10.00 volts. For single-ended AC outputs, Vref is a 20 volt P-P sinewave at 400 Hz. This allows the output to range from 0 to 20.00 volts P-P at 0 degrees phase shift relative to the reference and from 0 to 19.84 volts P-P at 180 degrees phase shift relative to the reference. For a differential output configuration, Vref is reduced to a 10 volt P-P sinewave at 400 Hz, providing the same output range as the single-ended configuration.

When the ICA is configured for analog output the DAC output is routed to the output buffers through M12. This is accomplished by addressing M12 with address 3. The logic for addressing M12 is in the ADCC section of the ICA.

The buffer amplifiers consist of M4, M5, M6 and their associated circuitry. These amplifiers must provide a very low output impedance to drive the subsystems and yet must be controllable so that they present essentially an open circuit when the channel is used for input. The requirement for low output impedance prevents the use of a conventional analog multiplexer to connect or disconnect the buffer from the input/output leads. A typical electronic analog switch has an on resistance of 100 ohms, which is far too large to add to the buffer output impedance. The circuit used solves the problem by utilizing complementary symmetry output buffers (Q1-Q2, Q3-Q4) inside the loop of an op-amp stage (M4A, M4B). The buffer stage is switched in and out of the op-amp loop by analog multiplexers M5, M6. The op-amp stages retain feedback in either mode through the multiplexer. Note that if the buffer is removed from the op-amp loop the only effect on the input/output lines is a small leakage current (the difference between the Icbo's of the NPN and PNP buffer transistors). When the buffer is activated the multiplexer switch resistance is inside the loop of the op-amp so that a very low output impedance is maintained. The complementary buffer is biased for class B operation and as such can contribute crossover distortion to high frequency AC waveforms. The op-amp selected has a slew rate of 13 v/μsec and at 400 Hz the crossover distortion is negligible.

Analog outputs can be either single-ended or differential depending upon the programming from the ADCC section. Each buffer amplifier (positive and negative) is individually controllable. As mentioned above, the ADCC logic automatically alters the output DAC reference so that the output range is always +/- 10 volts for either single-ended or a differential output modes.

3.1.5 DIGITAL OUTPUT PROCESSING

Digital output levels corresponding to the two binary logic states (one and zero) are programmed in the DAC's located in the reference generation section of the ICA. These levels are made available to the analog output buffers through the analog multiplexer M12. Each logic level can be independently programmed to a value in the range of +/- 10 volts. This allows interfacing to a wide range of logic families and easily implements a negative logic scheme. The logic in the ADCC section provides proper addressing to multiplexer M12 so that either address Ø (corresponding to the value programmed for a logic zero) or 1 (corresponding to the value programmed for a logic one) is selected based on the logic level programmed by the LM for each group channel. Note that the logic levels are the same for all channels within a group.

3.2 REFERENCE GENERATION

A block diagram of the reference generation portion of the ICA is shown in Figure 35. This portion generates the following reference signals: 400 Hz AC reference waveform, +5.00 volt DC(ADC reference), +10.00 vdc, +5.00 vdc, 20 vp-p ac, 10 vp-p ac(DAC references), HI, LO, and THRESHold levels used in processing digital signals in the SIO groups. Appendix B, Section 3-B contains a complete schematic and parts list for the reference generation section which will be referred to in the following discussion. The reference generation section contains portions which are shared by both groups and portions which are group specific. This commonality will be pointed out in the following sections.

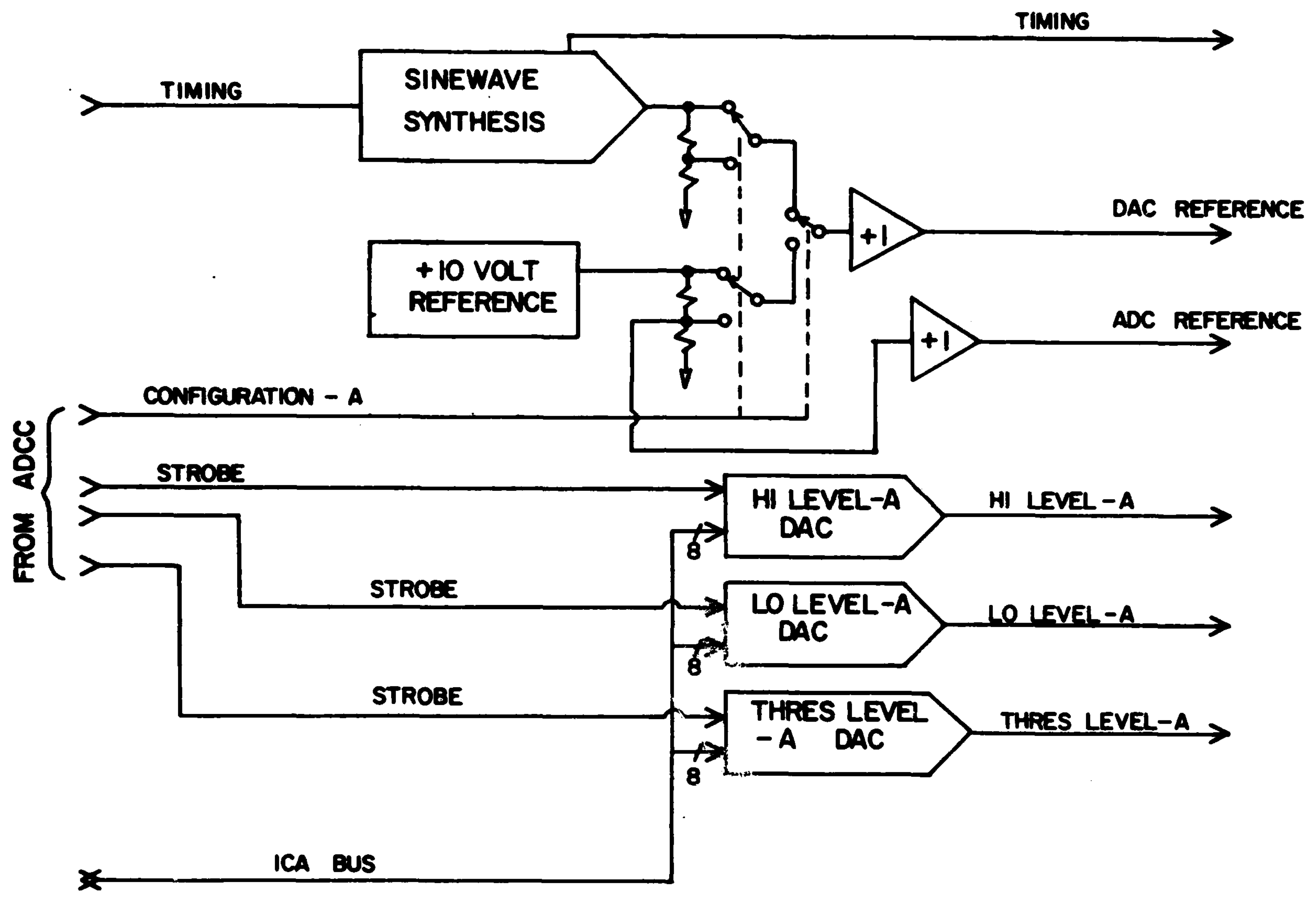


Figure 35 Block Diagram of a Reference Generation System

3.2.1 400 Hz AC REFERENCE

The 400 Hz AC reference signal is used to generate 400 Hz AC output signals and to provide timing to process 400 Hz AC input signals. It is derived from the LM master clock (1 MHz) by direct digital synthesis.

The basic timing is accomplished with counters M1, M2, and M3. M1 and M2 are used as divide by 5 counters, resulting in 40 KHz at the output of M2. The output of M3 is decoded in M5 at a count of 50 and reset resulting in a total division of 50×5×5 or 1250 and an 800 Hz timing signal, R800. M7 contains 50 8-bit samples of a sinewave. These samples, addressed at 0 through 49 within the ROM, cover one half cycle of the sinewave. The samples are applied to the reference DAC (M8) at a 40 KHz rate so that each half cycle is developed in 1/800 second. M10A and M9B, combined with timing through M14A and M14B, provide synchronous signal inversion so that a full 400 Hz sinewave results at the output of M9B. This signal is used internally as the AC reference for various DACs and is made available externally through the ACREF and ACRET lines in the ICA/Subsystem interface cable. Q1 and Q2 provide signal buffering for the AC reference.

Timing for the ADCs in the SIO channels is also generated in this section. The outputs from M3 are decoded in M6 at a count of 25 to provide the H/T timing signal. This signal results in the track and hold circuit entering the hold mode at the peak of the ICA reference. Assuming that there is no phase shift between the generation and the measurement of the reference, the ADC will digitize a peak value from the AC waveform. The signal +/- is also generated in this section (M14B, M18C). This signal is used in the SIO channel to synchronously invert AC signals.

The 400 Hz reference is buffered as described above and made available to the subsystem through connector J4. It is also used as the group analog output DAC reference when the group is configured for AC output. The full 20 vp-p reference is used for the single-ended configuration and a 10 vp-p reference is used for differential configurations.

3.2.2 +10.00 AND +5.00 vdc REFERENCES

M17A and its associated circuitry are used to generate the DC references. The output of M17A is a constant +10.00 volts, calibrated with R4. M17B provides a buffered +5.00 volts to be used as the reference voltage for the ADC's located in the SIO channels. The DACs in the SIO channels of Group A use as a reference the output of M15A. Group B DACs use the signal from M15B. These signals can be a constant +10.00 volts, a constant +5.00 volts, a 20 volt p-p 400 Hz sinewave or a 10 volt p-p 400 Hz sinewave depending upon the exact configuration programmed into the ICA. Single-ended DC outputs require a +10.00 volt DAC reference. Differential DC outputs use a +5.00 volt DAC reference so that the amplitude programming is the same for both single-ended and differential configurations. Single-ended AC outputs require a 20 volt p-p DAC reference while differential AC outputs use a 10 volt p-p DAC reference. The selection logic for Group A is implemented with M13A, M11A, M16A and M16B. The selection for Group B is implemented with M13B, M11B, M12A and M12B.

3.2.3 HI, LO, AND THRESHold LEVEL REFERENCES

U1 through U6 are used to generate and store the HI, LO, and THRESHold references used in groups A and B. Each Ui is a full four quadrant multiplying DAC which can be programmed directly from the ICA bus. The write strobes are generated in the ADCC circuitry described in Section

3.3. The DAC used for setting the digital THRESHold level uses the +10.00 volt reference and can generate a THRESHold value between + and - 10.00 volts. The DACs which hold the HI and LO digital output levels use D/A reference for the appropriate group. This reference is automatically set to +10.00 volts for single-ended DC output configurations, to +5.00 volts for differential DC output configurations, to a 20 volt p-p zero phase sine-wave for single-ended AC output configurations, and to a 10 volt p-p zero phase sinewave for differential AC output configurations. The programmed count in each of the DACs allows setting the HI and LO references to values within a factor of +1 to -1 of these values.

3.3 ADDRESS DECODING AND CONFIGURATION CONTROL

The address decoding and configuration control (ADCC) function is performed on a group basis. The following will discuss the operation of the ADCC section for Group A but is applicable to the Group B ADCC section as well. The block diagram for the ADCC function for one group is shown in Figure 36. Appendix B, Section 3-C contains a complete schematic and parts list for the ADCC section.

The ADCC function is divided into two basic operations: the decoding of addresses for the various addressable modules in the group and the control of the SIO configuration.

3.3.1 ADDRESS DECODING

The addressable modules associated with each group are:

1) The 7 DACs (4 for analog output values from the 4 SIO channels and 3 for the group HI, LO, and THRESHold values).

2) The 4 ADCs (one for each SIO channel).

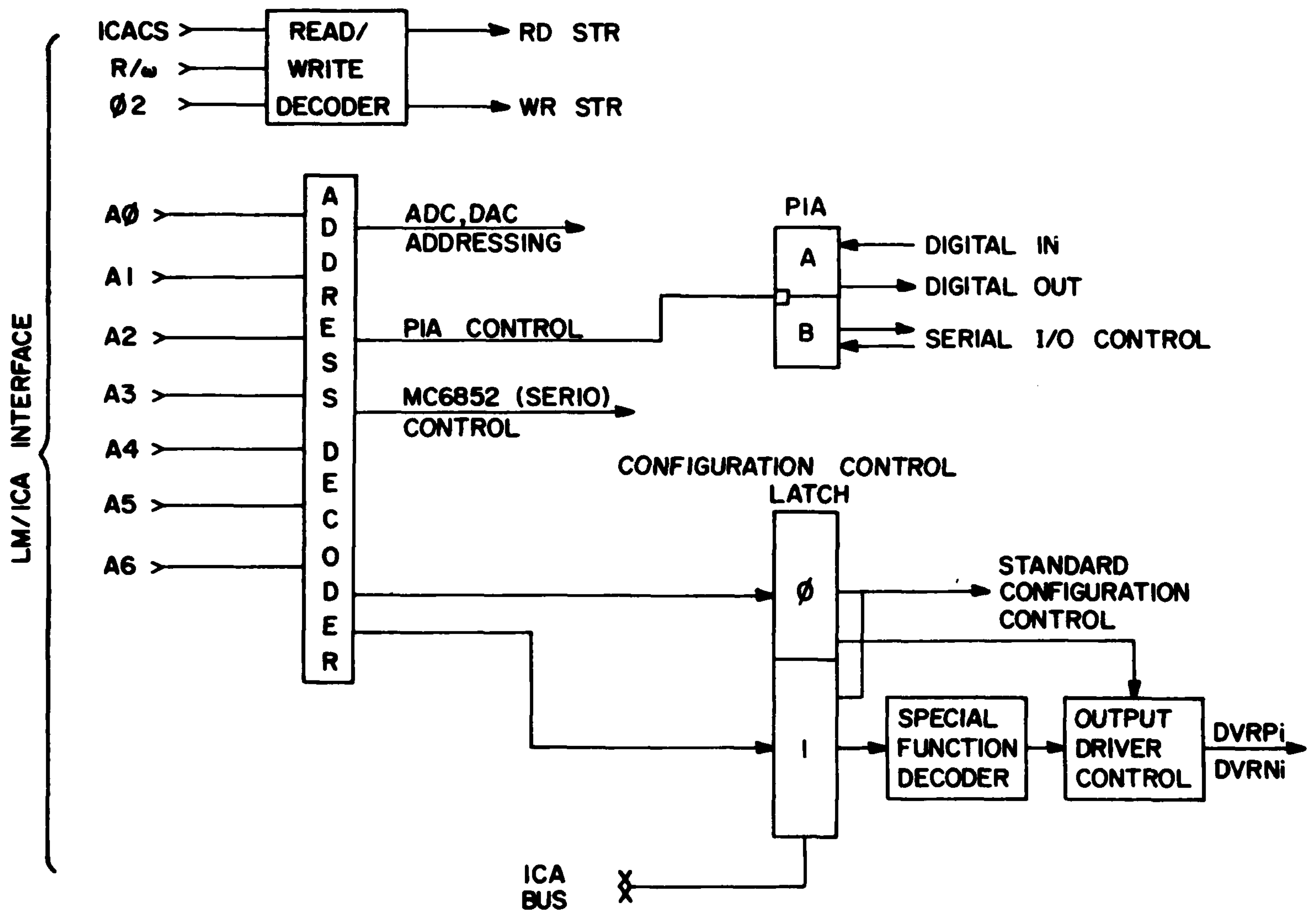


Figure 36 Block Diagram of the Address Decoding Section

3) The SIO configuration words (2 addresses).

4) The peripheral interface adaptor (PIA) (used for the 4 parallel digital lines of the group and to provide control lines associated with the serial I/O function, requires 4 addresses).

5) The synchronous serial data adapter (SSDA) (used for the serial I/O function, requires 2 addresses).

The above units require a total of 15 unique addresses (note that the 4 analog output DACs and the 4 analog input ADCs use the same 4 addresses, an LM write is directed to the appropriate channel DAC and an LM read is directed to the appropriate channel ADC). The address map for each group is presented in Figure 37.

M1 and M7 perform the basic address decoding function. M1 decodes relative addresses Ø through 7 while M7 decodes relative addresses 8 through 14. The logic of M4A-B, M5A-B, M6A-E, M10A, M10D, and M11A-B serves to select the address range in blocks of four addresses using the address lines A2 through A6 from the LM. Note that the only change required by Group B is the inclusion of an inverter in the A4 address line to displace the addresses by 16.

Devices are read by the LM by decoding the appropriate address and directing the read strobe to the appropriate register. The delays caused by the use of CMOS logic does not cause problems during reads since the data is taken from the ICA by the LM at the end of the machine cycle and the decoded read enable occurs near the beginning of the cycle. The setup times for the interface registers are met by using a 1 MHz clock for the LM clock. Devices are written into by decoding the appropriate address and directing the write strobe to the selected register. The delays caused by the use of CMOS logic are a problem since the data is to be taken from

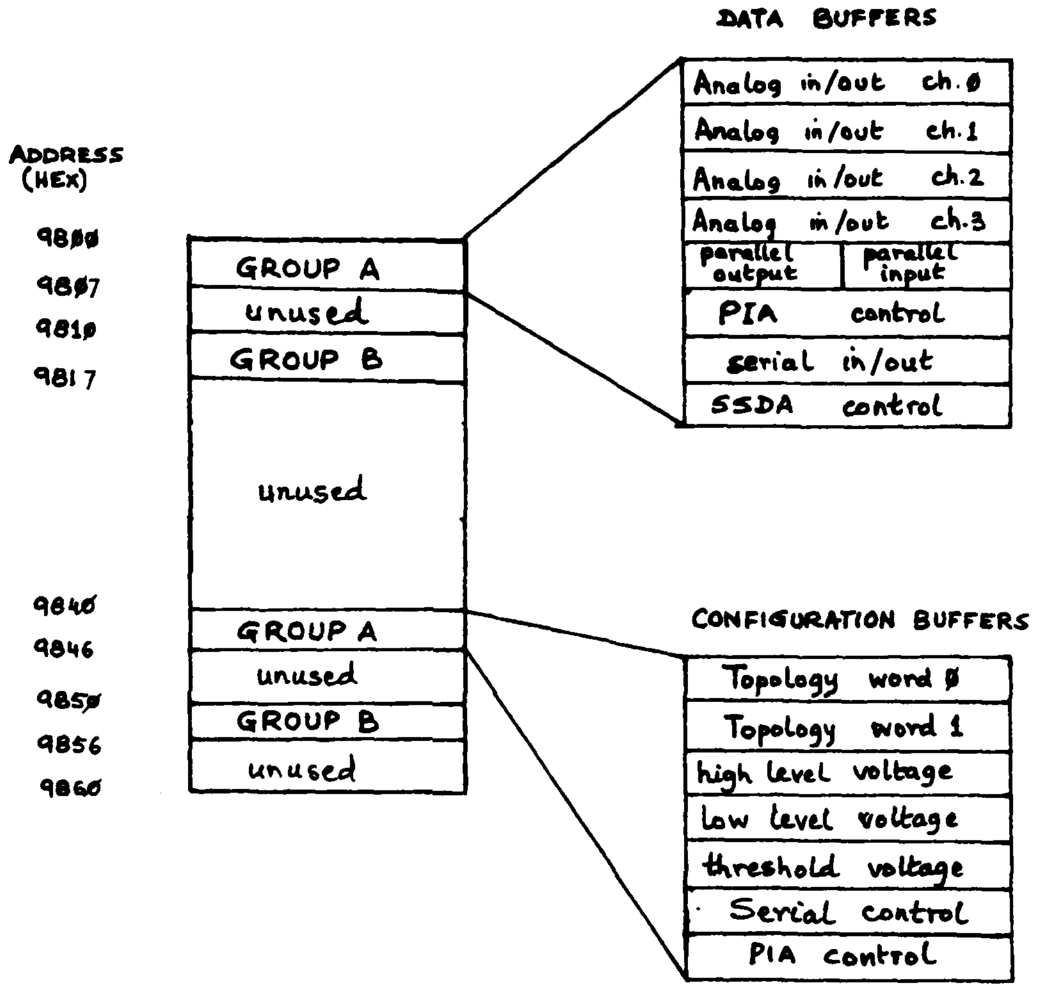


Figure 37 Memory Map of ICA Buffers

the data bus at the end of the machine cycle and the edge is delayed relative to the data. It is possible for the data on the LM bus to change before the decoded write edge propagates to the selected register. This problem is circumvented by the use of M29A which produces the write edge (ie, terminates the write strobe) before the LM cycle is finished.

3.3.2 CONFIGURATION CONTROL

The configuration words control the data type and direction through the SIO channels. Figure 38 contains a description of the bits used in each of the two configuration words. The standard configurations are those in which each channel of the SIO performs the same function. These include DC and AC analog input and output, as well as parallel digital input and output. The choice of single-ended versus differential input and output is determined by the programming of the input multiplexer address lines (INMUXØ-A, INMUX1-A and INMUX2-A) and the driver enable lines (DVRP and DVRN) respectively.

The non-standard SIO functions are those in which the channels are not individually programmed the same. These include the serial and synchro input and output modes. These special configurations are controlled by M16 and associated logic.

For synchro input, output S1 of M16 is high which results in inhibiting the drive enable lines to channels Ø, 1, and 2 of the group. The DVRP and DVRN lines can be applied to channel 3. This results in an AC reference being made available through channel 3 while channels Ø-2 are set up as single-ended input channels for the three legs of a synchro winding. Note that channel 3 is forced to a differential output configuration independently of the state of DVRN. This allows a full 40 vp-p

CONFIGURATION WORD Ø

AD	ACDC	DIV	DVP	DVN	IM2	IM1	IMØ

AD	-	Selects between analog (1) and digital (Ø).
ACDC	-	Selects between AC(1) and DC(Ø).
DIV	-	Thevenin source on (1) or off (Ø).
DVP	-	Positive voltage driver selected (1) or deselected (Ø).
DVN	-	Negative voltage driver selected (1) or deselected (Ø).
IMØ, IM1, IM2	-	Select one of eight input modes. Normal operations utilize single-ended (3) or differential (1) mode. The remaining six modes are used for test purposes.

CONFIGURATION WORD 1

DM3	DM2	DM1	DMØ	X	FLG	OEN	LF

DMØ-DM3	-	Select one of the special configurations: reset (Ø), test (15), serial-out (8), serial-in (4), synchro-out (2), and synchro-in (1).
FLG	-	Selects flag (1) or refresh (Ø) modes during serial input.
OEN	-	Voltage drivers on (1) or off (Ø).
LF	-	Selects between latched (1) and followed (Ø) modes during momentary discrete inputs.

Figure 38 ICA Configuration Words

differential output from channel 3 (if DVRN is programmed low so that the reference generation section maintains a full 20 vp-p reference) which makes it possible to drive a wide range of available synchros.

For synchro output the S2 output of M16 will be high. This allows channel 3 to be configured as a differential channel independently of DVRN. Normal synchro output would use channels ∅-2 (single-ended) to drive the three legs of the synchro control winding (referenced to the group signal return) and use channel 3 (differential) to drive the excitation winding as a two terminal floating load.

For serial input the four group channels are used as follows:

Channel ∅:	output (request/lockout)
Channel 1:	input (flag/acknowledge)
Channel 2:	output (serial clock)
Channel 3:	input (serial data)

This configuration is forced by the S4 output from M16 by inhibiting DVRN and DVRP for channels 1 and 3.

For serial output the four group channels are used as follows:

Channel ∅:	output (request/lockout)
Channel 1:	input (flag/acknowledge)
Channel 2:	output (serial clock)
Channel 3:	output (serial data)

This configuration is forced by the S8 output from M16 by inhibiting DVRN and DVRP for channel 1.

3.4 SERIAL I/O DESIGN

A block diagram of the Serial I/O section is shown in Figure 39. This section contains the logic necessary to implement the serial I/O function including parallel-to-serial and serial-to-parallel conversion, FIFO buffering, and parity checking. Appendix B, Section 3-D contains a complete schematic and parts list for the Serial I/O section.

The function is performed by a combination of discrete logic and a single LSI programmable Synchronous Serial Data Adapter (SSDA), the Motorola MC6852.

The MC6852 circuit provides the functions of FIFO buffering, parallel-to-serial conversion, serial-to-parallel conversion, and parity checking. The control of the chip is performed by external logic and internal programming. The MC6852 contains a total of 11 internal registers including a 3 byte receive FIFO, a 3 byte transmit FIFO, 3 control registers, a status register, and a sync code register. The ICA function uses all of these registers with the exception of the sync code register.

The following discussion will reference the schematic diagram and parts list in Appendix B, Section 3-D. The discussion will focus on the SERIO function for Group A but applies as well to the SERIO circuitry in Group B.

The bit rate is selected through M14 at 200 Kbps, 40 Kbps, 20 Kbps, or 10 Kbps. The address to M14 is programmed via the CLKRTØA and CLKRT1A lines from the Group A ADCC logic. The clock rate selection is as follows:

Bit Rate	CLKRTØA	CLKRT1A
200 Kbps	1	1
40 Kbps	1	Ø
20 Kbps	Ø	1
10 Kbps	Ø	Ø

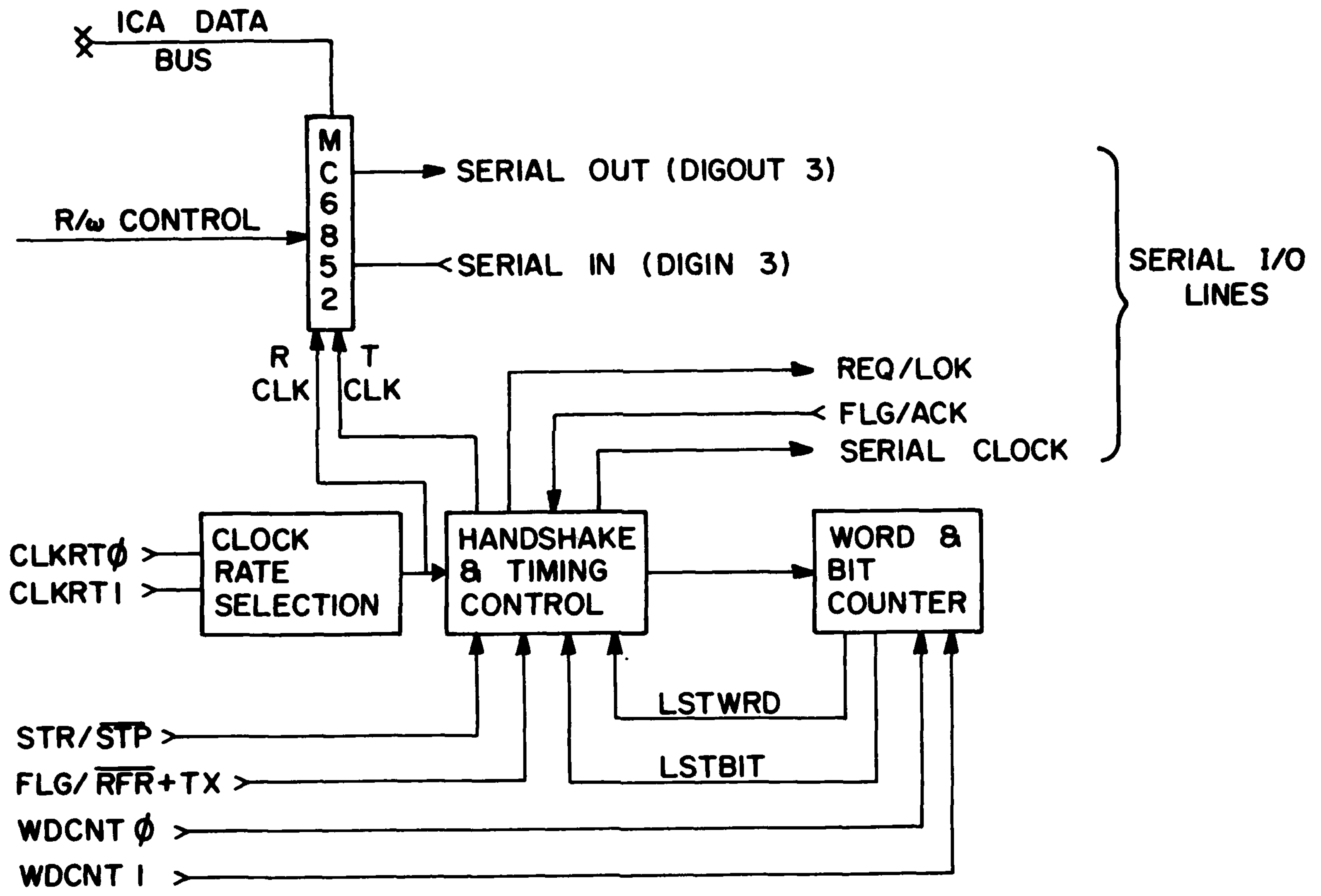


Figure 39 Block Diagram of the Serial I/O Section

The number of bytes of data to be transferred is programmed through the WDCNTØA and WDCNT1A lines. These lines originate in the Group A ADCC logic and are applied to M11 in the SERIO section.

Words	WDCNTØA	WDCNT1A
1	Ø	Ø
2	Ø	1
3	1	Ø

The SERIO circuitry can operate in three modes:

1) Output Mode

2) Input, Refresh Mode

3) Input, Flag Mode .

These modes are discussed below.

3.4.1 SERIAL OUTPUT

The serial I/O circuitry transmits synchronous serial digital output data in this mode. The circuit can be programmed to output 1, 2, or 3 bytes of information during a transfer. The transfer is initiated by the LM by raising the STR/$\overline{\text{STP}}$ line. This raises the REQ/LOK line to the subsystem. If the subsystem is able to accept data it responds by raising the FLAG/ACKnowledge line (DIGIN1) to the ICA. This initiates the transmission of (N*9) clock pulses to the subsystem where N is the number of words programmed for transmission. The last bit transferred is the parity bit for the last word. This bit is checked in the subsystem at the last falling edge of the clock and the parity information is used by the subsystem to set the final state of the FLAG/ACK line. The LM inspects the End of Transmission (EOT) bit to determine when the transfer has been completed. The LM can then determine the state of the FLAG/ACK line and finally lower the REQ/LOK line to the subsystem. If an error occurred the LM can take appropriate action. The serial data originates at pin 6 of M1 and is buffered through the appropriate signal I/O channel to the subsystem.

3.4.2 SERIAL INPUT-REFRESH MODE

The refresh mode is one of two input modes provided by the ICA. This mode allows input to take place under control of the ICA through requests on the REQ/LOK line. The data is transferred to the ICA and is received by the FIFO memory in the MC6852. The process is similar to the output mode in that (N*9) clock pulses are provided by the SERIO circuit and the transmission is not completed until the LM has determined if a parity error has occurred. Such error is detected by the LM by reading the received words and their associated parity bits out of the FIFO memory of the MC6852.

3.4.3 SERIAL INPUT-FLAG MODE

The flag mode allows the interfaced subsystem to initiate a data transfer to the ICA. The SERIO circuitry must first be programmed for the expected number of data bytes and the clock rate to be used with the transfer. The transfer process is initiated when the subsystem raises the FLAG/ACK (DIGIN1) line to the ICA. This action causes the REQ/LOK line to the subsystem to be raised in response and allows the programmed number of clock cycles to be generated and sent to the subsystem. The clock is frozen in the middle of the last programmed cycle so that the LM can check on the parity bit state for each of the received words in the FIFO memory of the MC6852. If the transmission is correct, the REQ/LOK line is lowered to the subsystem by lowering the ERR line in the SERIO circuit. The last clock edge is then produced through the CLKLOW line from the ICA and the subsystem terminates the transmission cycle. If an error occurs during reception, the CLKLOW line is lowered while the REQ/LOK line is still high and the subsystem will be ready to retransmit the message.

The parameters and control bits required for serial transmission/reception are specified in the serial control byte of the ICA configuration buffers. This byte is set up in the data register of a Peripheral Interface Adapter (PIA) and is illustrated in detail in Figure 40.

SERIAL CONTROL

SS	ERR	FLGR	WDC1	WDCØ	CLK1	CLKØ	EOT

SS - This bit controls the STR/$\overline{\text{STP}}$ line of the SERIO circuitry. It is used to initiate (1) or terminate (Ø) the serial reception/transmission.

ERR - This bit controls the ERR line in the SERIO circuitry.

FLGR - This bit controls the serial input mode: Flag (1) or refresh (Ø).

WDCØ, WDC1 - These bits control the number of bytes of data to be received/transmitted.

CLKØ, CLK1 - These bits control the clock rate used for serial reception/transmission.

EOT - This bit indicates if the serial transmission is complete (1) or not (Ø).

Note: The CLKLOW line in the SERIO circuitry is controlled by the CB2 peripheral output line of this PIA. CLKLOW is normally low and is set high to provide the last clock pulse, after which it is brought low again.

Figure 40 Serial Control Byte

SECTION 4

SUBSYSTEM INFORMATION CHANNEL

The *Subsystem Information Channel* (SIC) is a distributed memory system that stores information pertaining to devices connected to a computer system. The information for each device is stored in an electronic nameplate physically located on that device. Thus it becomes simple to change a system's configuration, since at initialization the processor may interrogate the SIC to determine which devices are present and establish the interface requirement for each device. The information stored in a device's nameplate includes:

- Device identification,
- Interface characteristics,
- Data conversion programs,
- Calibration programs,
- Diagnostic programs,
- Failure/maintenance records written when in previous use.

The SIC consists of two major types of components (Figure 41):

- Electronic Nameplate (NP): The information storage unit located on the peripheral device,
- Nameplate Interface Controller (NIC): An interface module which resides in the processor chassis and enables communication between the processor and nameplates.

Many subsystems consist of several modules, each with specific interface requirements. In this case each module may be equipped with an electronic nameplate containing information specific to that module. All

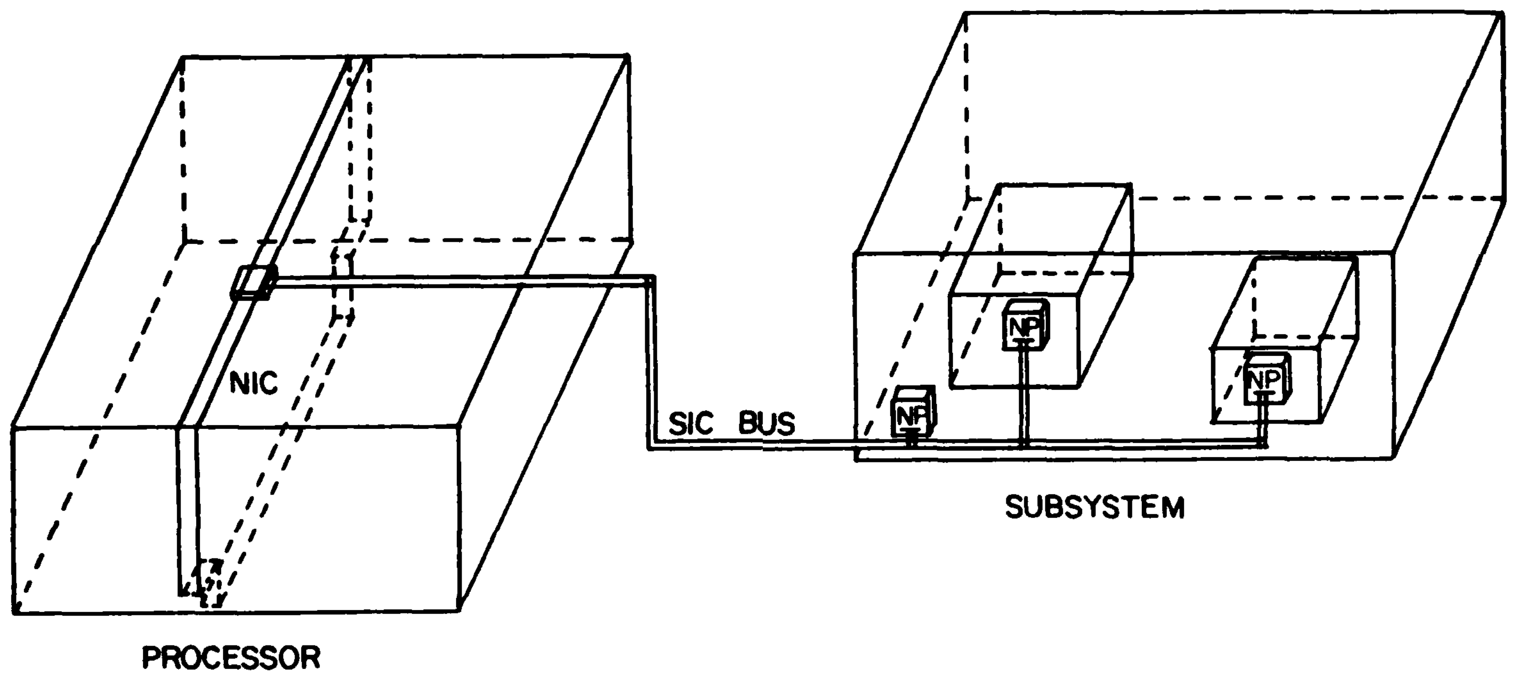


Figure 41 The Subsystem Information Channel

lectronic nameplates within a subsystem are "daisy-chained" along an SIC bus.

The Subsystem Information Channel is viewed as a distributed memory by programs running in the processor. Programs may "read" the nameplate's memory through the SIC by using SIC commands.

The following section gives an explanation of how to use the SIC and later sections discuss the design of the parts of the Subsystem Information Channel.

4.1 USE OF A SUBSYSTEM INFORMATION CHANNEL

This section describes the control and communication protocol utilized by the subsystem information channel to access any of its electronic nameplates. The communication protocol and its use are presented first. Use of the control, status and data registers of the nameplate interface controller is presented next. A structure for storing the information residing in an electronic nameplate is presented at the end of this section.

4.1.1 ELECTRONIC NAMEPLATE COMMANDS

Control of a nameplate by a processor is accomplished through the use of electronic nameplate commands. The nameplate receiving the command sends a specific response to the processor acknowledging that the required action has been performed. The nameplate commands may be grouped into four functional categories:

- **Nameplate selection commands,**
- **Read memory commands,**
- **Write memory commands,**
- **Nameplate diagnostic commands.**

4.1.1.1 Nameplate Selection Commands

The nameplate selection commands are used to establish processor communication with one of several nameplates. The two commands "select level ∅ NP" and "select next NP" allow a program to sequentially select nameplates on the SIC bus. This procedure starts with the nameplate nearest to the processor which, by position, is assigned level ∅. Once a nameplate is selected, it may be assigned an eight bit address which may be used for subsequent random nameplate selection. This is accomplished by the commands "assign NP address" and "select NP by address". During initialization the sequential selection mode is used to assign nameplate addresses. Thereafter a particular nameplate is selected usually by its address.

The identity of the nameplate selected can be determined at any time through the use of the command "read selected NP's address". Absence of a response to this command indicates that no nameplate is selected. Each nameplate sets its address to an invalid address (-1) whenever it is reset. A nameplate can be reset by the processor via the nameplate interface controller or whenever the nameplate's power or clock is absent.

Only one nameplate can be selected at a time. If a nameplate is selected and a "select level ∅ NP" command is issued, the selected nameplate will deselect and the level ∅ nameplate will become selected and respond to the command. If a nameplate is selected and a "select next NP" is issued, the presently selected nameplate deselects, and the next nameplate on the SIC bus (further from the processor) becomes selected and responds to the command. Note that the "select next NP" command has no effect if no nameplate is selected. If a nameplate is selected and it is desired to select another nameplate using the "select NP by address" command, the first nameplate must be deselected using the command, "deselect NP", and then the desired nameplate may be selected.

A bit in the nameplate status byte (the first byte of every response) indicates if the responding nameplate is the last nameplate on the SIC bus (i.e. the farthest from the processor). If the last nameplate is selected and a "select next NP" command is issued, the last nameplate will deselect and no nameplate will be selected thus no response will be issued.

4.1.1.2 Read Memory Command

Once an electronic nameplate is selected, its memory may be read. The nameplate's memory consists of read only memory used for storage of subsystem interfacing information and read/write memory used for storage of subsystem performance records. The command "read selected NP's memory" allows the processor to transfer up to 256 bytes of data from the nameplate's read only or read/write memory. This is accomplished by sending with the command the starting address and number of bytes to be transferred. If more than 256 bytes are required, additional read memory commands may be issued each with a new starting address. Only if the starting address and ending address (i.e. starting address + number of bytes requested) are within the nameplate's memory boundaries will the data transfer request be honored. Otherwise the nameplate's response will indicate an invalid address status.

4.1.1.3 Write Memory Commands

Information may be stored in a nameplate's read/write memory through the use of the "write NP memory" command. For the sake of security, a write command to a nameplate will not be honored unless it is preceeded by the "NP write enable" command.

Data written into a nameplate's read/write memory is stored in records. Each record consists of a 16 byte block. The starting address of each record, sent with the write command, must start on a 16 byte boundary (i.e. the last hexadecimal digit of the address must be zero). The last two bytes of each record are used to store a record checksum and a record terminator. These bytes are inserted by the nameplate logic when the record is written. Thus the maximum number of data bytes that can be written into a record is fourteen. The most significant bit of the first byte in each record is utilized as a record validity indicator. Modifying this bit is the last operation performed in response to a write command. This bit when set to zero indicates that the record is properly written.

Each nameplate may contain sixteen such written records at any time. These records are written in non-volatile memory (i.e. the nameplate may be powered down and the written information will be retained). The starting address of the first unwritten record space (i.e. the unused record space with the lowest address) may be obtained by using the command "next available record address". An address of zero returned in response to this command indicates that the nameplate's read/write memory is full.

A nameplate's read/write memory can be erased through the use of the "erase read/write memory" command. The erase command is in the write category of commands and therefore must be preceeded by the "NP write enable" command. The present implementation of the electronic nameplate does not support the actual erasure of memory. Receipt of the erase command is acknowledged by lighting the nameplate's erase indicator. The nameplate's read/write memory must be removed from the nameplate board and exposed to ultra-violet light for fifteen minutes in order to be erased.

Execution of the write (or erase) command requires approximately one second. The nameplate will acknowledge receipt of these commands immediately and will proceed to execute the write (or erase) action. During this time only the select, deselect commands and the "read selected NP address" command will be accepted and executed. Other commands will be rejected and the response will indicate "NP busy". However, if it is desired to terminate a write (or erase) prior to its completion, the command "abort selected NP" may be used.

The nameplate should be disabled from writing (or erasing) as soon as the desired write (or erase) has been completed. This action will avoid accidental modification of the read/write memory. The write disable is accomplished through the use of "NP write enable" command with the enable/disable flag not equal to one.

4.1.1.4 NP Diagnostic Commands

The last group of commands cause the nameplate to run a self-test program. When the command "run NP diagnostic" is issued, the selected nameplate will respond immediately and start execution of its diagnostic routine. The diagnostic routine requires approximately 1/4 second of execution. During diagnostic execution only the select, deselect commands and the "read selected NP's address" command will be accepted. Other commands will be rejected with a status of "NP busy". Execution of the diagnostic routine may be prematurely terminated with the "abort selected NP" command. When the diagnostic is completed, the results may be read by issuing the command "read NP diagnostic results". Figure 42 illustrates

First Result Byte

b7	b6	b5	b4	b3	b2	b1	bØ
DCG	ROME	Ø	RECE	Ø	RAME	TIME	Ø

DGC:	Diagnostic completed
ROME:	ROM data errors
RECE:	Read/Write records in error
RAME:	RAM errors
TIME:	Timer errors

Second Result Byte

b7	b6	b5	b4	b3	b2	b1	bØ
ROM4	ROM3	ROM2	ROM1	—FR	EE—	REC—	——

ROM4:	ROM number 4, address (58ØØ-5FFF)$_{16}$, data errors
ROM3:	ROM number 3, address (5ØØØ-57FF)$_{16}$, data errors
ROM2:	ROM number 2, address (48ØØ-4FFF)$_{16}$, data errors
ROM1:	ROM number 1, address (4ØØØ-47FF)$_{16}$, data errors
FREE REC:	Number of read/write records still available

NOTE: A one in the associated bit position indicates the condition is true.

Figure 42 NP Diagnostic Result Data Bytes

the format in which the results are returned.

4.1.1.5 Command and Response Structure

The structure of all valid nameplate commands is presented in Table 16. The same table also identifies the format of the response to each command. Commands and responses have different numbers of bytes depending on the amount of information required by the action requested. Figures 43a and 43b show the message structure of various commands and associated responses in terms of their byte contents. Note that the nameplate status is always the first byte of the response. When an error is encountered in the execution of a command the second (and last) data byte of the response will diagnose the cause of the error. The structure of the status bytes is presented in Figures 44a and 44b.

4.1.2 NAMEPLATE INTERFACE CONTROLLER REGISTERS

The nameplate interface controller is the controller circuit, residing in the processor chassis, through which communication with nameplates takes place. As viewed by the processor, the nameplate interface controller appears as a set of memory mapped control, status and data registers.

4.1.2.1 SIC Status and Control Registers

One of these registers, the SIC status register, reflects the status of the subsystem information channel as shown in Figure 45a . Bits 7, 4 and 3 reflect the signal level of specific SIC hardware control lines. Bits 2, 1 and Ø are outputs of the SIC clock divider circuit. Consecutive reads of the status register should yield a changed value for bits 2, 1 and Ø if the SIC clock is functioning correctly. This value changes approximately every 13.1 microseconds.

Table 16

NAMEPLATE COMMANDS

COMMAND		RESPONSE	
Instruction	Data	Status	Data
1. Select level ∅ NP	N/A	NP status	NP address
2. Select next NP	N/A	NP status	Newly selected NP address
3. Select NP by address	NP address	NP status	NP address
4. Deselect NP	N/A	NP status	NP address
5. Assign NP address	Address to be assigned	NP status	NP address
6. Read selected NP's address	N/A	NP status	NP address
7. Read selected NP's memory	Number of memory bytes wanted (0=256), starting memory address	NP status	Number of memory bytes, starting memory address, memory data
8. Write enable/ disable	Enable/disable flag (Enable=1)	NP status	NP address
9. Write memory data	Number of data bytes, starting memory address, Data to be written	NP status	Starting memory address
10. Next available address to be written	N/A	NP status	Address of next available write memory record
11. Erase read/ write memory	N/A	NP status	NP address
12. Run NP diagnostic	N/A	NP status	NP address
13. Read NP diagnostic results	N/A	NP status	Diagnostic results
14. Abort selected NP	N/A	NP status	NP address

Command Number								
1, 2, 4, 6 10, 11, 12, 13, 14	CMD	CKSUM						
3, 5, 8	CMD	DATA	CKSUM					
7	CMD	BYTE RQ	MEM ADRH	MEM ADRL	CKSUM			
9	CMD	BYTE CT	MEM ADRH	MEM ADRL	DATA	• • •	DATA	CKSUM

Figure 43a SIC Command Byte Structure

Command Number								
1, 2, 3, 4, 5, 6, 8, 11, 12, 14	STATUS	NP ADR	CKSUM					
NP busy	STATUS	NP ADR	CKSUM					
any error	STATUS	ERSTAT	CKSUM					
9, 10	STATUS	MEM ADRH	MEM ADRL	CKSUM				
7	STATUS	BYTE CT	MEM ADRH	MEM ADRL	DATA	• • •	DATA	CKSUM
13	STATUS	DIAG1	DIAG2	CKSUM				

Figure 43b SIC Command Response Byte Structure

b7	b6	b5	b4	b3	b2	b1	bØ
CER	NPB	LNP	WRE	DLA	WED	DT1	DTØ

CER:	Command error
NPB:	NP is busy
LNP:	This NP is the last NP on the SIC bus
WRE:	Error occurred on last write
DLA:	Deselect acknowledge
WED:	Write enabled/disabled flag (enabled=1)
DT1, DTØ:	Indicated type of data in response as follows:

CER	DT1	DTØ	Response Data
Ø	Ø	Ø	Nameplate Address
Ø	Ø	1	Diagnostic Results
Ø	1	Ø	Memory Address Only
Ø	1	1	Memory Address and Memory Data
1	X	X	Error Diagnostic Byte

NOTE: A one in the associated bit position indicates condition is true.

Figure 44a Nameplate Status Byte

b7	b6	b5	b4	b3	b2	b1	bØ
INC	INA	INW	CER	INS	Ø	Ø	Ø

INC:	Invalid command
INA:	Invalid memory address (either starting or ending address)
INW:	Invalid write (erase) request, write not enabled
CER:	Communication error
INS:	Another NP requested while this NP is still selected

NOTE: A one in the associated bit position indicates the condition is true

Figure 44b Error Diagnostic Byte

b7	b6	b5	b4	b3	b2	b1	bØ
PWR	Ø	Ø	NPR	R/C	CL2	CL1	CLØ

Address:
X X X 2

PWR: SIC power status (1=SIC power not present)
NPR: NP reset line level
R/C: RESP/CMD line level
CL2, CL1, CLØ: SIC clock status bits

NOTE: A one in the associated bit position indicates the condition is true

Figure 45a SIC Status Register

b7	b6	b5	b4	b3	b2	b1	bØ
PWO	Ø	1	1	NPR	1	1	PIE

Address:
X X X 3

PWO: Power outage (or SIC disconnection) since last read of SIC status register
NPR: NP reset control
PIE: Power outage (or SIC disconnection) interrupt enable

NOTE: A one in the associated bit position indicates the condition is true

Figure 45b SIC Control Register

Figure 45b illustrates the SIC control register. This register is used primarily for initializing the SIC status register and for resetting all nameplates. The simultaneous reset of all nameplates is accomplished by setting (to one) bit 3 (NPR) of the control register. As long as the NPR bit is set, all nameplates will remain in the reset state. To enable the nameplates after a reset is performed, the NPR bit should be reset (to zero). The processor should wait approximately 40 milliseconds after removing the reset signal before attempting to access a nameplate. This delay is required to allow the nameplates to initialize.

The PWO bit will be set in the SIC control register whenever the subsystem information channel loses power as a result of power supply malfunctions or breaks in the SIC bus (e.g. on reconfiguration). The PWO bit will stay set even if power returns to the SIC bus. This bit will be cleared when the SIC status register is read. The PWR bit (bit 7) of the SIC status register indicates the status of the SIC power at the time of the read. Thus if power goes down and returns between readings of the SIC status register, the PWR bit of the SIC status register will be zero in both readings. However the PWO bit of the SIC control register will be set to one indicating that a power loss has occurred. The SIC control register should always be read before reading the SIC status register since reading the SIC status register clears the PWO bit of the SIC control register. An interrupt to the processor may be generated when the PWO bit is set to one. The PIE bit (bit Ø) of the control register controls the enabling and disabling of this interrupt.

The SIC status and control registers should be initialized in the following manner:

i) write $(3A)_{16}$ to the SIC control register,

ii) write $(\emptyset\emptyset)_{16}$ to the SIC status register,

iii) write $(36)_{16}$ to the SIC control register,

iv) read the SIC status register.

After this procedure is performed the SIC status and control registers' bit functions are as indicated in Figures 45a and b.

Additional information on the functions of each bit in the SIC control and status registers is presented in the nameplate interface controller hardware schematics (Appendix B, Section 4-A and the data sheets on the Motorola MC6821 peripheral interface adapter.

4.1.2.2 SIC Communication Registers

The sending commands and receiving responses from nameplates is accomplished with the aid of two registers: the SIC communication control/status register and the SIC communication data register. Figures 46a and 46b depict the format of these registers.

The SIC communication control/status accepts controls when written and provides communication status when read. The R/C bit (bit 6) of the SIC communication control register controls the direction of the data on the SIC serial data bus. The SIC is enabled to send a command to the nameplates by writing a zero into the R/C bit. The nameplates are enabled to respond to a command when the processor writes a one to this control bit. Two other important bits, RIE and TIE, are used to enable receiver buffer full and transmitter buffer empty interrupts respectively. To initialize the SIC communications, the value $(\emptyset 3)_{16}$ should be written into the communication control register.

Control Register

b7	b6	b5	b4	b3	b2	b1	bØ
RIE	R/C	TIE	1	Ø	1	Ø	1

Address:
X X X 4
(write only)

RIE:	Receiver data available interrupt enable
R/C:	RESP/CMD enable control (1=RESP)
TIE:	Transmitter empty interrupt enable

NOTE: To reset communications, write: Ø3.

Status Register

b7	b6	b5	b4	b3	b2	b1	bØ
IRQ	Ø	OVR	FE	R/C	Ø	TXE	RDA

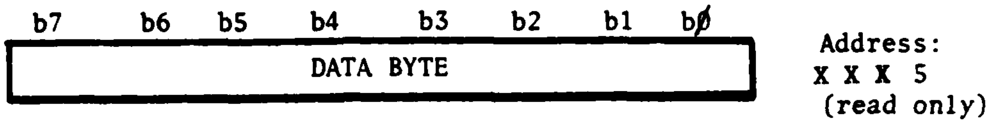
Address:
X X X 4
(read only)

IRQ:	Interrupt present
OVR:	Receiver data overrun error
FE:	Receiver data framing error
R/C:	RESP/CMD enable status (1=RESP)
TXE:	Transmitter data register empty
RDA:	Receiver data available

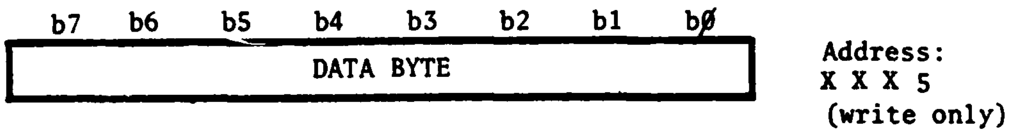
NOTE: A one in the associated bit position indicates the condition is true

Figure 46a SIC Communication Control/Status Register

Receive Register



Transmit Register



NOTE: Data is in 8-bit bytes, no parity.

Figure 46b SIC Communication Data Register

Reading the SIC communication control/status register will yield the SIC communication status as indicated in Figure 46a .

Command or data to be transmitted to a nameplate must be written into the SIC communication data register (Figures 46b). Transmission to the nameplates will only take place if the R/C control bit is reset (i.e., command enabled). The response from a nameplate is read from the SIC communication data register (Figure 46b). No response will be received unless the R/C control bit is set to one (i.e., response enabled). The processor should delay approximately 2.1 milliseconds after transmitting the last byte of a command before enabling the response (i.e. setting the R/C control bit to one). This action will assure that all bytes sent are received by the nameplates.

A more detailed explanation of the functions of each bit in the SIC communication registers is provided in the nameplate interface controller hardware schematics (Appendix B, Section 4-A) and the data sheets on the Motorola MC6850 asynchronous interface adapter.

4.1.3 NAMEPLATE'S DATA STRUCTURE

The electronic nameplate can be viewed as a component of a distributed memory system. The distributed memory (i.e., the nameplate's memory) is used to store information related to the subsystem on which the memory is attached. The design of the electronic nameplate places no restriction on the data structure of information stored in the nameplate's read only memory. The nameplate's read/write memory is used for storing subsystem performance information. The read/write memory is organized in terms of 16 byte records. Fourteen of the bytes in a record may be used for data storage and are not restricted in format. A restriction is placed on the most significant bit of the first byte of each record. This bit will be set to zero by the nameplate logic if the record is written correctly.

The nameplate's memory format allows a variety of data structures to be used for storage of subsystem information. In order to facilitate information retrieval, the organization of subsystem data within a nameplate should have a well defined structure. The rest of this section will discuss the nameplate's subsystem information storage structure for implementation of a subsystem information channel.

Each nameplate will have a directory in read only memory identifying all information structures present in the nameplate. The format of this directory is illustrated in Figure 47. The directory of any nameplate starts at a fixed (known) memory address. The different information modules which a nameplate may have in read only memory include:

- Subsystem identification,
- Subsystem interfacing characteristics,
- Subsystem I/O routines,
- Subsystem diagnostic/calibration routine.

The presence of each of the above modules in a nameplate's memory will be identified with a corresponding entry in the directory.

The initial segment of each information module contains a header. This header provides a physical description of the module. The information in a header includes:

- Identification of the module,
- Size of the module,
- Address of the first executable instruction,
- Checksum for the module.

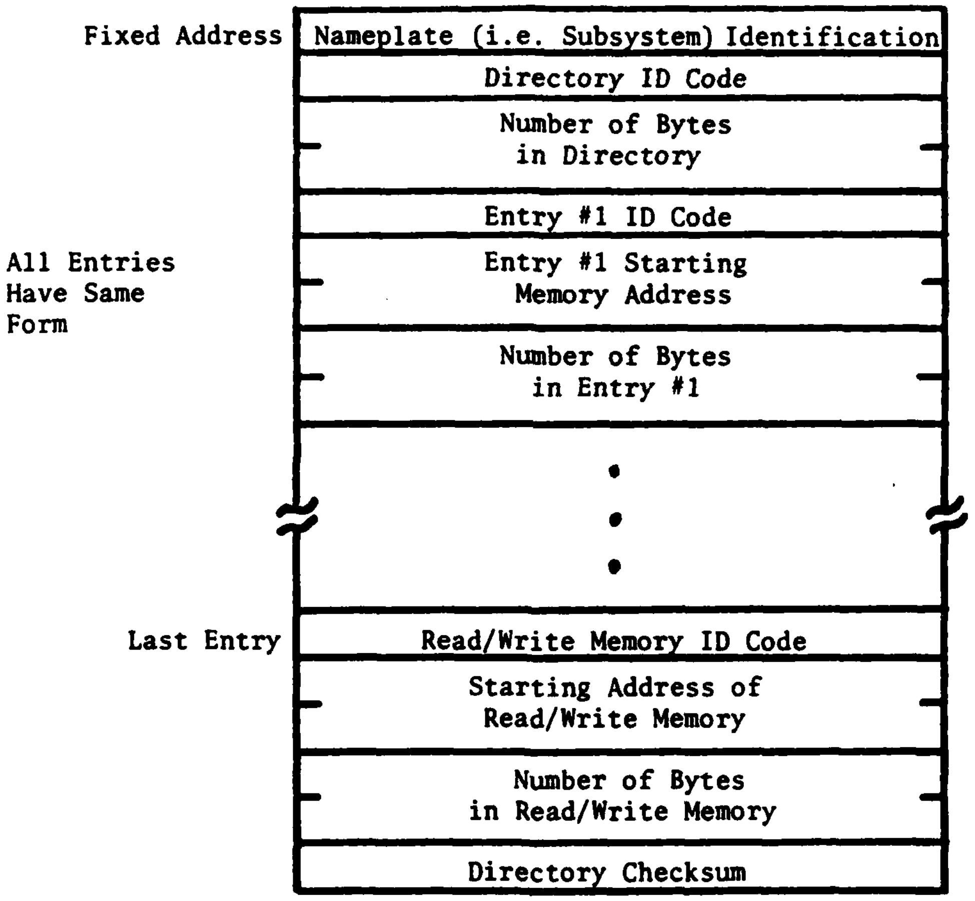

Fixed Address
Nameplate (i.e. Subsystem) Identification
Directory ID Code
Number of Bytes in Directory
Entry #1 ID Code
All Entries Have Same Form
Entry #1 Starting Memory Address
Number of Bytes in Entry #1
Last Entry
Read/Write Memory ID Code
Starting Address of Read/Write Memory
Number of Bytes in Read/Write Memory
Directory Checksum

Figure 47 Nameplate Directory Structure

The last entry in a nameplate's directory contains the starting address and size of the read/write memory area. The format of a subsystem performance record written in the read/write area is shown in Figure 48. The first byte is an identifier for the type of data in the record. Typical record types include:

- Subsystem failure reports,
- Subsystem repair reports,
- Subsystem calibration results,
- Subsystem diagnostic results.

The eight unspecified bytes in a record may be used to store information relevant to the action causing the recording (i.e. record type).

4.2 ELECTRONIC NAMEPLATE DESIGN

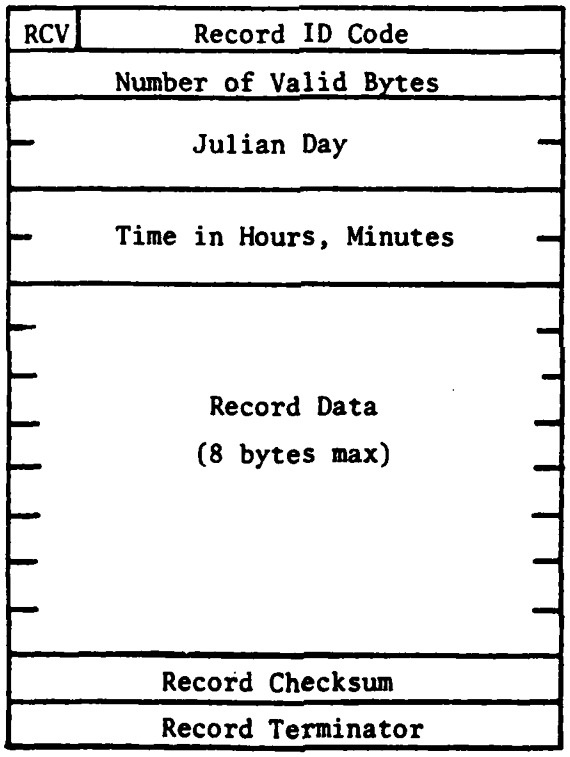
4.2.1 HARDWARE DESIGN

The design of electronic nameplates for the subsystem information channel is based on a Motorola MC6801L1 processor. The electrical specifications for the nameplate are presented in Table 17. One should observe that power for all nameplates is supplied by the SIC bus. Therefore the SIC power supplies must be sized to handle the maximum number of nameplates for a given application.

Theory of Operation

A description of the nameplate's operation can be made in terms of the block diagram shown in Figure 49. The detailed schematic drawings for the nameplate are presented in Appendix B, Section 4-B.

The majority of the logic functions required from the electronic



RCV: Record Validity Bit

Figure 48 Subsystem Performance Record

Table 17

ELECTRONIC NAMEPLATE SPECIFICATIONS

CHARACTERISTIC	SPECIFICATIONS
Power Requirements	+5Vdc ±5% @ 1.2 Amps +25Vdc ±5% @ 50ma
Input Signals	TTL voltage compatible
Output Signals	TTL voltage compatible
Operating Frequency	614.4 K Hz
Operating Temperature	0 to 70°C
External Interfaces	SIC Bus

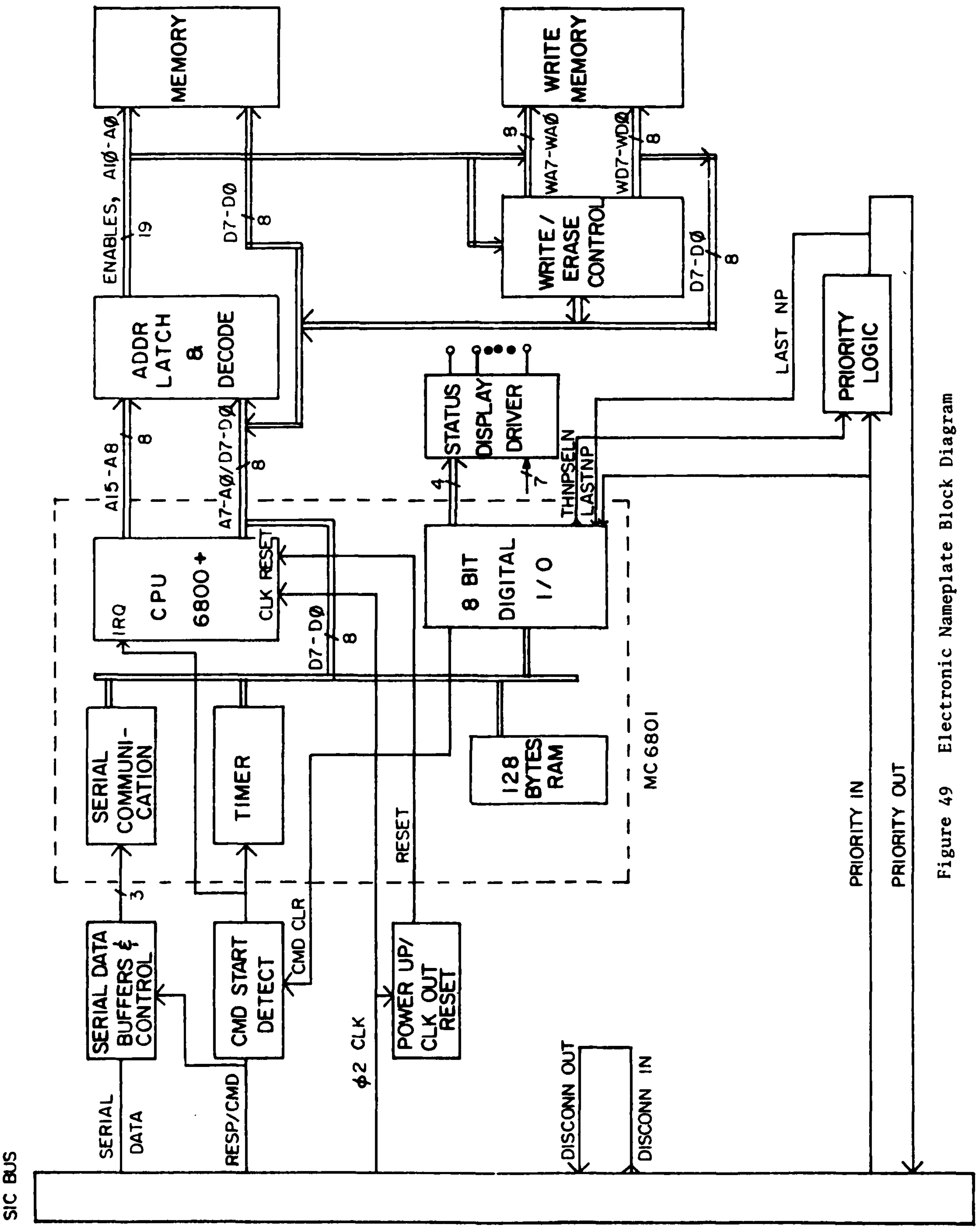


Figure 49 Electronic Nameplate Block Diagram

nameplate are performed by a Motorola MC6801L1 processor. This processor is configured in its expanded multiplexed mode of operation and includes the following operational elements:

- 6800+ CPU: Motorola 6800 processor with an enhanced instruction set,
- 128 bytes of RAM memory,
- serial transmitter/receiver communication device,
- free running timer which can output software controlled pulses and also measure times between input transitions,
- 8 bits of digital I/O.

Detailed specifications for the processor are provided by Motorola documentation on the MC6801L1.

All external signals required by the nameplate are provided by the SIC bus. These signals are shown on the left side of Figure 49. The function of each major component identified in the nameplate block diagram will be described in the remainder of this section.

Reset Circuit

When power is turned on or the SIC clock recovers from an interruption, this circuit causes the nameplate logic to reset. The reset signal goes false approximately 30 milliseconds after the power and clock are both present.

Serial Data Buffers

This logic circuit uses the RESP/CMD SIC bus signal to control the direction of data flow to or from the SIC simplex serial data line.

Command Detection

This circuit detects when the RESP/CMD signal goes low indicating the start of a new command from the system processor. This negative going edge is latched, causing an interrupt in the 6801 CPU. The 6801 CPU clears the interrupt through one of its 8 digital I/O lines.

Address Latch and Decoding

The expanded multiplexed mode of operation (mode 1) of the 6801 multiplexes the lower 8 bits of address with the corresponding data bits. Therefore the 8 lower order address bits must be demultiplexed with an external latch. Address lines A15 through A5 are then applied to a decoding ROM that generates the chip select signals for various nameplate memory components and control devices.

Write/Erase Control

This logic provides the capability to write 256 bytes of data on an EPROM. This logic contains an 8 bit latch for holding the address and an 8 bit latch for holding the data to be written. It also provides the write control required to tri-state the EPROM from the nameplate's data and address buses during programming. The write control also gates the MC6801 timer's 50 millisecond pulse and switches the +25 volt power as required to program the EPROM.

The EPROM implementation of the read/write memory does not permit electrical erasure. The erase control utilizes the 6801 timer to generate a one second erase pulse. This pulse, in the present implementation, lights an LED which is used for command verification.

Priority Logic

The priority logic "ANDs" the PRIORITY IN signal from the SIC bus with the signal THNPSELN (this nameplate is selected), an output signal from the 6801 digital I/O port, to form the SIC bus signal PRIORITY OUTN. The SIC bus connects to the nameplate in such a way that the PRIORITY OUTN signal of one nameplate becomes the PRIORITY IN signal of the next nameplate farther away from the system processor. If a nameplate is selected, its THNPSELN signal is low. Thus the PRIORITY IN signals of all nameplates farther away from the processor are low.

A line of the 6801's I/O port is used to allow the nameplate's software to test the value of its PRIORITY IN signal. When a nameplate's PRIORITY IN signal is low, it indicates that some nameplate closer to the system processor (i.e. a higher priority nameplate) is selected. The nameplate sensing a low PRIORITY IN has lower priority and therefore must function accordingly.

Another line of the 6801's I/O port, the LAST NP signal, is checked by the nameplate's software to determine if it is the last nameplate on the SIC bus (i.e. the farthest from the system processor). The nameplate's PRIORITY OUTN signal passes through a current limiting resistor before it is applied to the SIC bus. The LAST NP signal is connected to the PRIORITY OUTN signal on the SIC bus side of this resistor. For all nameplates that are not the last on the bus, the signal LAST NP follows the signal PRIORITY OUTN. The SIC bus terminator, placed after the last nameplate on the SIC bus, jumpers the PRIORITY OUTN signal to the SIC +5 volt power line. Thus in the last nameplate, the signal LAST NP will be high even when the inputs to the priority logic circuit (THNPSELN and

PRIORITY IN) dictate the signal should be low. The nameplate's software, by monitoring the inputs to the priority logic and this LAST NP signal, can determine if it is the last nameplate on the SIC bus.

Status Display

These buffers are used to drive LED displays that indicate the nameplate status. Some of the displays indicate the internal status of the 6801 and its software. The remaining LEDs are driven by important hardware signals. Table 18 lists the status identified through LED displays.

SIC Bus

The SIC bus provides the medium for a processor to communicate with one or more nameplates. Control of the bus by the processor is achieved with the nameplate interface controller residing in the processor. The SIC bus also supplies power and the clock for the nameplates. When multiple nameplates are used they are "daisy chained" along the SIC bus as illustrated in Figure 50.

The SIC bus must be properly terminated at the last nameplate. This termination consists of two jumpers connecting the bus signals BPRIORITY and BDISCONN to the +5 volt power line. The BPRIORITY jumper is used to indicate this nameplate is the last one on the bus. The BDISCONN jumper causes the corresponding bus line to be held high at the nameplate interface controller. This will remain true until the SIC "daisy chain" is opened. The nameplate interface controller uses this signal to detect when a nameplate (and corresponding subsystem) has been replaced or reconfigured. Table 19 identifies the signals comprising the SIC bus. Appendix B, Section 4-B contains detailed information on the cable implementation.

Table 18

NP STATUS DISPLAY

LED NUMBER*	STATUS DISPLAYED
1	NP is busy
2	This NP is selected
3	SIC RESP/CMD bus line level (CMD=ON)
4	SIC SERIAL DATA bus line level
5	EPROM write strobe
6	EPROM simulated erase strobe
7	SIC PRIORITY IN bus line level (High=on)
8	NP Diagnostic is executing

*LED #1 is the left most LED, LED #8 is farthest to the right.

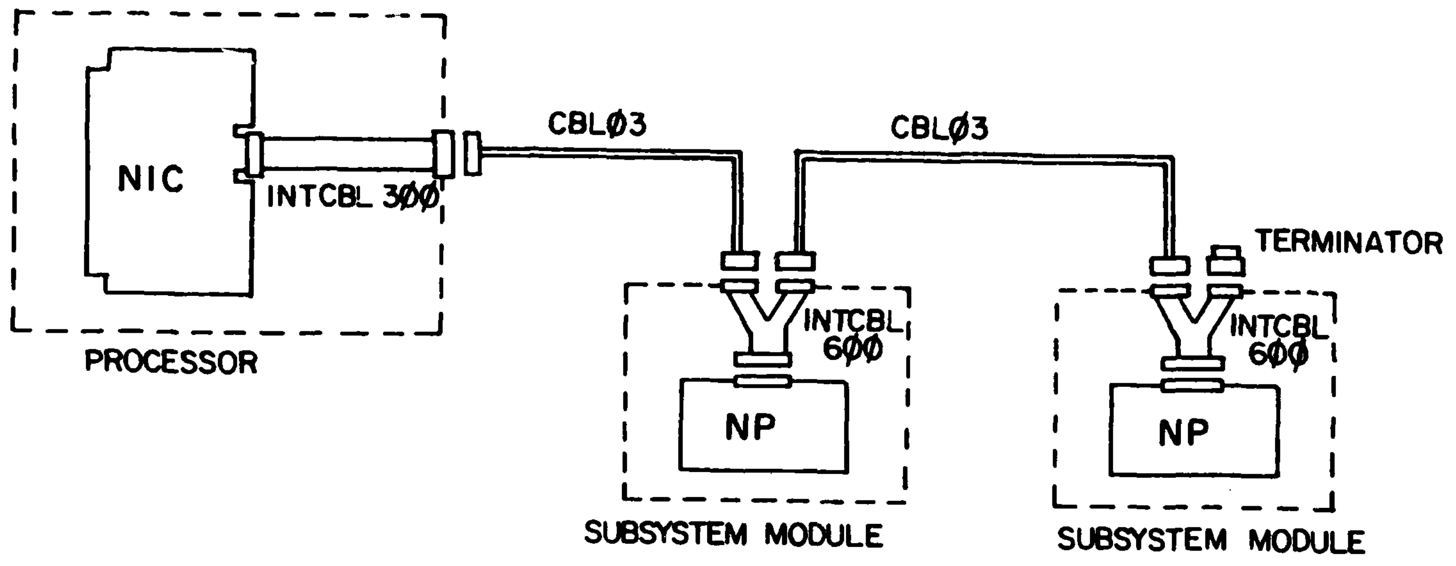


Figure 50 SIC Bus Connections

Table 19

SIC BUS SIGNALS

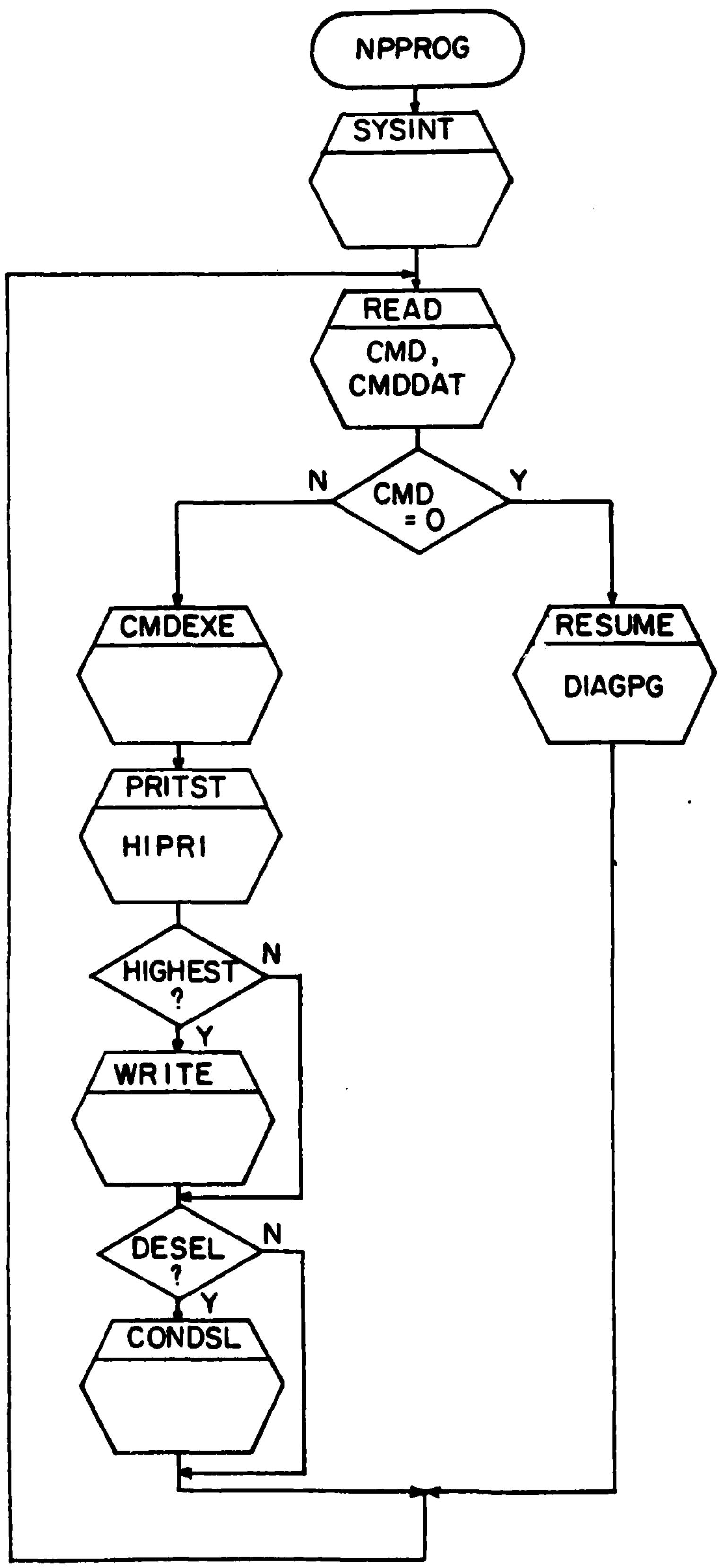
SIGNAL NAME	FUNCTION
BØ2 CLK	SIC system clock -2.4576 MHz
BPRIORITYN	Nameplate priority status line
BSERIAL DATA	Serial data communication line
BRESP/CMD	Response/command enable line
BDISCONN	SIC reconfiguration detection line
GROUND	SIC ground (2 lines)
+5V	SIC +5Vdc logic power (2 lines)
+25V	SIC +25Vdc EPROM programming power line

4.2.2 SOFTWARE DESIGN

The software of a nameplate consists primarily of one program, entitled NPPROG, which executes as a continuous loop. Whenever a nameplate is requested to run its diagnostic, the program DIAGPG runs "concurrently" with the program NPPROG. In addition to the two programs, several interrupt routines are used to perform functions requiring fast response.

A flowchart of the nameplate program NPPROG is shown in Figure 51. Table 20 gives a brief explanation of the subroutines called by the program. Normal execution starts when the nameplate is reset causing processing to vector to the start of NPPROG. The subroutine SYSINT initializes the nameplate's variables and devices. The READ subroutine will wait for the communication interrupt handlers to receive a command and its associated data. The wait by the READ subroutine is terminated when a read complete flag is set.

The CMDEXE routine decodes the command and calls the proper internal subroutine to execute the command and set up the proper data for the response. Certain commands (write data, erase, and run diagnostic) require more execution time than others. The subroutines of CMDEXE associated with long commands merely set up data and flags necessary for the interrupt handlers (or diagnostic program) to perform the actual execution. The response to these commands is used to acknowledge their receipt and the initiation of their execution. Thus the time from the reception of the last of a command to the start of the response will always be the same. Except for select, deselect and "read NP address" commands which are always accepted and executed, all other commands received while a long command is being executed will be rejected by the subroutines of CMDEXE. Such commands when received will result in a response with a "NP busy" status.



COMMENTS

RESET Interrupt Vector

Initialize System Variables

Get Received Command and Data; Loop if DIAGPG not Running or Read not complete

Diagnostic Program Active?

Resume Diagnostic Program
or
Execute Command and set up Response

Check if this NP has the highest priority of selected NP's

If highest priority, send response

Deselect here on Deselect Command after response sent.

Figure 51 Nameplate Main Program

Table 20

NAMEPLATE PROGRAM DIRECTORY

ROUTINE	DESCRIPTION
NPPROG	Main nameplate program, controls execution of other routines
SYSINT	Initialization routine
READ	Transfers data from the receiver interrupt handler's buffer to a command buffer
CMDEXE	Decodes and executes commands Checks if any NP of higher priority is selected
WRITE	Sends response to the SIC bus
CONDSL	Deselects this NP if "deselect nameplate" command was received
RESUME	Resumes suspended diagnostic program
DIAGPG	NP diagnostic program
SUSPEN	Suspends the diagnostic program

The PRITST subroutine is called by the main program to check the PRIORITY IN line. If this line is low, a higher priority nameplate (i.e. one closer to the system processor) is selected in which case this nameplate must not send its response. The nameplate's timer is used to perform a wait after the RESP/CMD line goes high and prior to the checking of the PRIORITY IN line. This is to assure that all nameplates are synchronized and thus the PRIORITY IN line is stable.

If this nameplate has highest priority (i.e. it is selected and its PRIORITY IN line is high) the subroutine WRITE proceeds to send the response set up by the CMDEXE subroutine.

If the command received is a deselect command, the subroutine CONDSL will then proceed to deselect this nameplate. This assures that when the nameplate is responding with a status of "deselect acknowledge" it remains selected until the response is completely sent.

The program then returns to the READ subroutine to check for another command. This loop executes continuously unless the nameplate's diagnostic program DIAGPG is requested to execute. A flowchart for the diagnostic program is presented in Figure 52. Through the use of the subroutine RESUME and SUSPEN, the diagnostic program and NPPROG are able to run "concurrently" as co-routines. Each of the programs NPPROG and DIAGPG use these subroutines to relinquish CPU control to the other at appropriate places in their execution. DIAGPG will execute for a time then suspend itself allowing NPPROG to check for any commands received. If there are none, NPPROG will resume the diagnostic program. As long as there are commands received that need to be processed the DIAGPG program must wait.

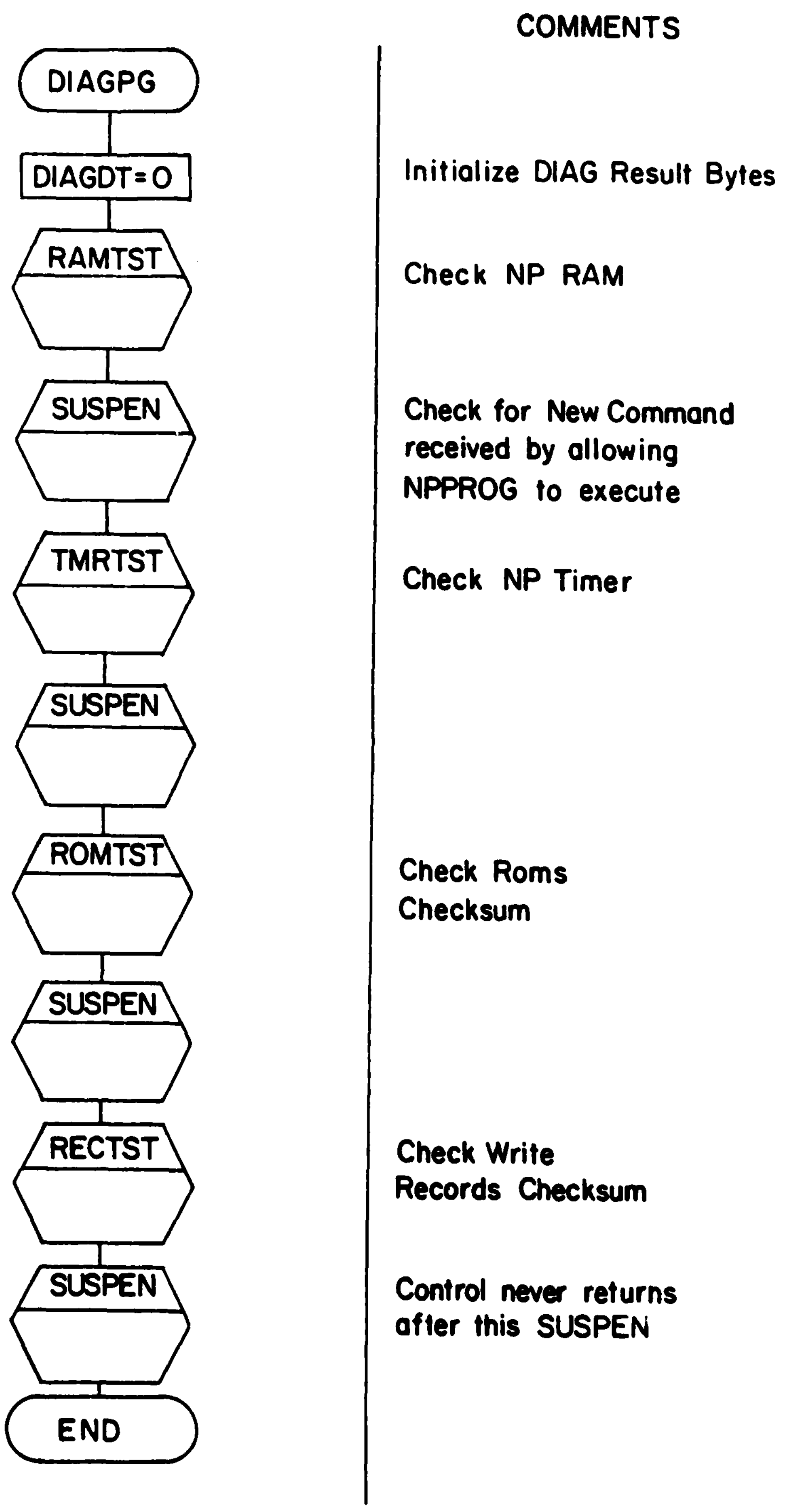

COMMENTS
DIAGPG
DIAGDT=0
Initialize DIAG Result Bytes
RAMTST
Check NP RAM
SUSPEN
Check for New Command received by allowing NPPROG to execute
TMRTST
Check NP Timer
SUSPEN
ROMTST
Check Roms Checksum
SUSPEN
RECTST
Check Write Records Checksum
SUSPEN
Control never returns after this SUSPEN
END

Figure 52 Nameplate Diagnostic Program

Interrupt handling routines are used in the nameplate to perform functions which are "transparent" to the main program. These routines are listed in Table 21. All these routines process data buffers and set flags to indicate their status of execution to the main program. Upon completion, these routines return to the program at the point of interruption so that the flow of the main program remains the same.

Communication between various routines is accomplished through common areas. Tables 22, 23 and 24 identify the common blocks of variables and their functions. Not all routines need access to all common variables, so the common area is divided into two local common areas and a global common area.

Detailed program descriptions including flowcharts for the nameplate software are presented in Appendix C, Section 4-A.

4.3 PROCESSOR INTERFACE TO THE SIC

In order to facilitate interaction of programs running in the processor with electronic nameplates installed in the SIC, an interface consisting of a hardware nameplate interface controller board and a software SIC handler has been developed. The SIC handler consists of a set of software routines designed to perform all control and monitoring actions required for communication.

4.3.1 NAMEPLATE INTERFACE CONTROLLER DESIGN

The nameplate interface controller (NIC) provides the hardware resources required to interface the processor with the SIC bus. A block diagram for the nameplate interface controller is presented in Figure 53. Operational and interface specifications for the NIC are presented in

Table 21

NAMEPLATE INTERRUPT HANDLERS

NAME	INTERRUPT SOURCE	DESCRIPTION
NPPROG	RESETN	Start execution of the main program
CMDSTR	IRQN from RESP/CMD line	Initializes serial communication to receive a command
RXDAT	RDA of serial receiver	Stores incoming data bytes in temporary buffer
CMDSTP	Timer Input Capture from RESP/CMD line	Terminates command reception
WRTERA	Timer Output Compare	Writes data to EPROM, or simulates memory erase.

Table 22

NPCOMM - NAMEPLATE GLOBAL COMMON

NAME	SIZE(bytes)	DESCRIPTION
THNPSL	1	This NP selected flag
NPSTAT	1	NP status byte
ERSTAT	1	Error reason status
NPADDR	1	Assigned NP address
NPBUSY	1	NP busy flag
NPSYCT	2	NP synchronization count
CMD	1	NP command being processed
CMDDAT	18	Data for command
CMDCNT	1	Number of Data bytes received with a command
TXCNT	2	Number of bytes in response
TXDATA	4	First 4 bytes of response
DIAGAD	2	Diagnostic program resume address
DIADDT	2	Diagnostic result data
DIAGSV	4	Diagnostic save buffer

Table 23

RDCOMM - READ COMMAND LOCAL COMMON

NAME	SIZE(bytes)	DESCRIPTION
RDCOMP	1	Read complete flag
RDSTAT	1	Status of received data
RDBYCT	1	Number of bytes received
READBF	20	Temporary buffer for received data

Table 24

WRTCOM - WRITE MEMORY LOCAL COMMON

NAME	SIZE(bytes)	DESCRIPTION
WRTBUF	15	Data to be written
RECADR	2	Starting address of record to be written
WRTPTR	1	Pointer into WRTBUF
DATTYP	1	Type of data to be written; Erase memory, write memory data, write valid record bit
WRTFLG	1	Write enable/disable flag
ERASCT	1	Count of time interrupts for erase
WRTTRI	1	Write re-try count

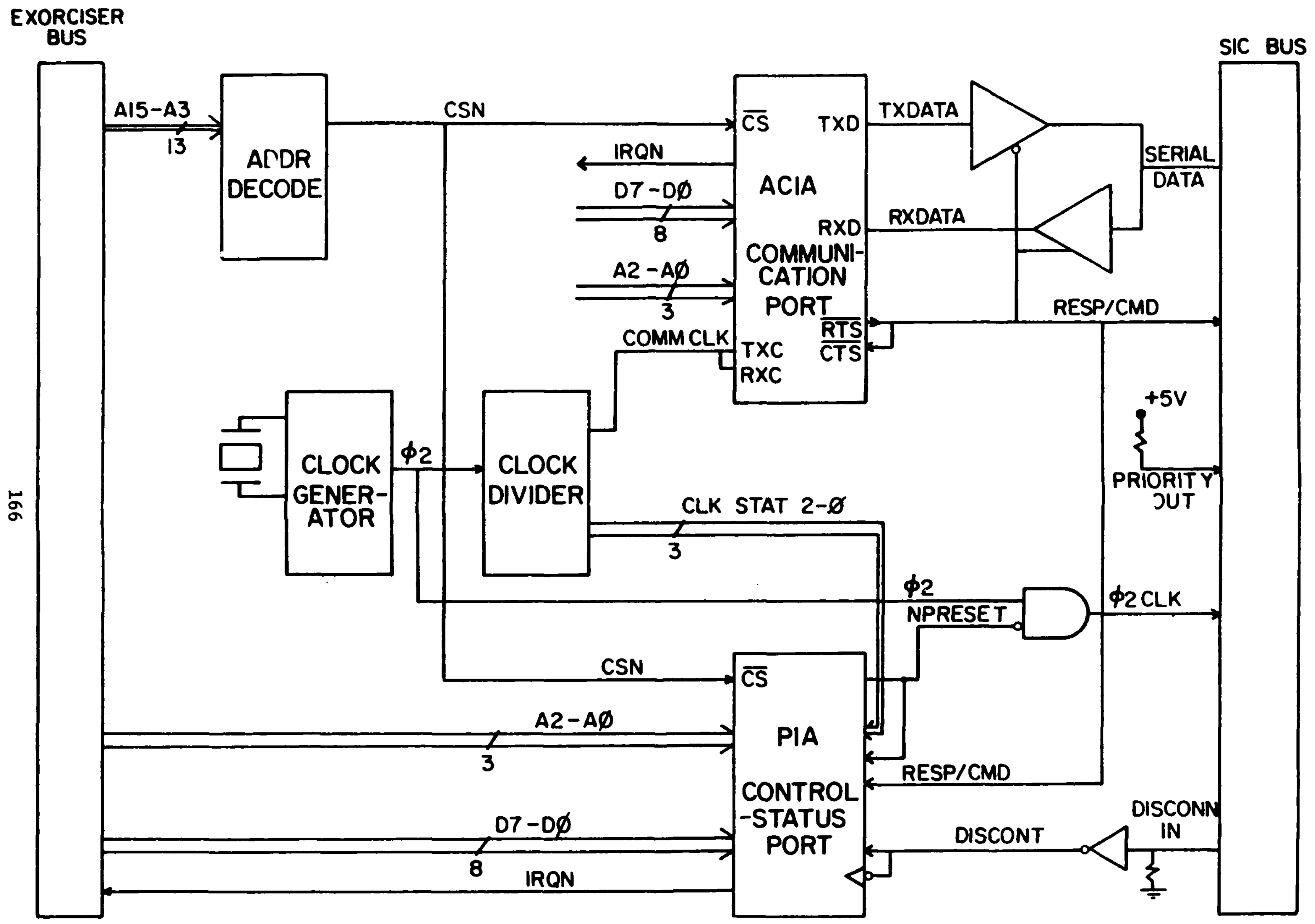


Figure 53 Nameplate Interface Controller Block Diagram

Tables 25 and 26. The NIC may be used in any processor's chassis utilizing the Motorola EXORciser bus. The address space for the NIC consists of 8 bytes (addresses XXXØ through XXX7). Of these addresses, only addresses XXX2 through XXX5 are actually used. The symbol X designates a hexadecimal number (Ø through F) in the address field that is user selectable through switches.

Theory of Operation

This section provides a description of the nameplate interface controller block diagram shown in Figure 53. Appendix B, Section 4-A contains the detailed schematic diagrams for this controller.

Address Decoding and EXORciser Bus Buffers

The NIC is built on a MEX68USM Motorola Universal Support Module which provides the address decoding and buffering of the EXORciser bus signals. EXORciser bus address signals A15-A3 are decoded to generate a chip select signal CSN. This signal and address lines A2, A1 and AØ are used to specify the NIC register requested. Switches S1, S2, and S3 are used to select the upper three hexadecimal digits of the address. Switch S4 should always be set to zero. One is referred to the MEX68USM user's guide for the location of these switches and for detailed information pertaining to the address decoding and bus buffering.

Serial Communications

An LSI device, the Motorola MC6850 (ACIA), is used to convert an 8 bit byte of parallel data into a 10 bit serial data stream (on transmission) and to convert 10 bits of serial data into 8 bit parallel data (on reception). The serial data is transmitted and received at a rate of 4800

Table 25

NAMEPLATE INTERFACE CONTROLLER SPECIFICATIONS

CHARACTERISTIC	SPECIFICATION
Power Requirements	+5Vdc +5% @ 2.1 Amps +12Vdc +5% @ 0.5 Amps -12Vdc +5% @ 0.4 Amps
Input Signals	TTL voltage compatible
Output Signals	TTL voltage compatible
Operating Temperature	0 to 70°C
External Interfaces	Motorola EXORciser Bus [1] SIC Bus [2]

[1] Refer to MC6800 EXORciser User's Guide.

[2] Refer to Appendix B, Section 4-B.

Table 26

NIC REGISTERS[1]

ADDRESS [2]	FUNCTION
X X X 2	SIC Status Register
X X X 3	SIC Control Register
X X X 4	SIC Communication Control Register
X X X 5	SIC Communication Data Register

1 See Section 4.1.2 of this document for a full explanation of the use of these registers.

2 X means this hexadecimal digit is switch selectable (Ø-F).

baud. The ACIA also provides a modem control signal which is used to implement the RESP/CMD signal of the SIC bus. This signal is used to control the operation of tri-state buffers which direct the flow of data on the simplex serial data line of the SIC bus. One is referred to Motorola's documentation on the MC6850 for more detailed information on its operation and use.

Clock Generator and Divider

A 4.9152 MHz crystal and a clock generator integrated circuit are used to generate a TTL compatible clock signal, called $\phi 2$ CLK, of frequency 2.4576 MHz for the SIC bus. This frequency is then divided by 32 to create a ×16 clock for the serial communication ACIA. Also three divider outputs (1/128, 1/64, 1/32) are input to the SIC status register so that consecutive reads of the register may be used to determine if the clock circuit is functioning.

SIC Control and Status

An LSI device, the Motorola MC6821 (PIA), is used as the SIC status and control registers. One of the PIA's 8 bit data ports is configured as an input port and is used to monitor several important SIC signal lines. This data port is called the SIC status register. The PIA's control register associated with this data port, called the SIC control register, controls the modes of operation of the SIC and generates a reset signal for the SIC bus. The reset signal kills the clock signal going into the SIC bus and causes the nameplate to reset.

LM Timer Circuit

Another function residing on the nameplate interface controller card is the LM timer circuit. This circuit divides the LM's 1 MHz system clock by 10^4 to obtain a 100 Hz signal. This signal triggers an interrupt through the PIA on the nameplate interface controller every 10 milliseconds.

4.3.2 SIC HANDLER DESIGN

A handler was developed to simplify the interface between tasks running in the LM and the control registers in the nameplate interface controller. A calling task requests a service (such as "load the nameplate's directory") from the handler through an argument list. The handler translates this request into a sequence of nameplate commands required to perform the function. The handler also performs all the control and timing actions required by the SIC communication protocol as described in the preceeding sections. Use and theory of operation of this SIC handler are presented in Section 2.3.5.

SECTION 5

LINK MANAGER

5.1 DESIGN OBJECTIVES

The Link Manager (LMG) Simulator described in this document is designed to meet the following objectives:

1. Verification of Link Module (LM) operation, demonstration of all functions proposed for the prototype LM.

2. Ability to exercise subsystems interfaced through the Interface Configuration Adapter (ICA), thus verifying the operation of the prototype ICA.

3. Different operational options:

   (a) 'real-time' simulation with LM - fast command stream from a disk file. As soon as the execution of one command is finished, the next command from the disk file will be given to the LM.

   (b) 'manual' operation of LM - commands input from a CRT manually. The results can be seen in steps at the CRT.

   (c) 'combined' operation - 'real-time' and 'manual' operation can be switched back and forth.

   (d) 'repetitive' operation - operations can be looped at high speed for exercising and debugging.

4. Facilities to easily set up a simulation or repetitive test, or set up a command stream for 'real-time' operation.

5. Ease of use, with simplified procedures for operating the LMG simulator.

6. Ease of modification. Software will be modular so as to facilitate future modifications to the LMG simulator.

5.2 SOFTWARE DESIGN OVERVIEW

5.2.1 GENERAL FEATURES

In order to satisfy the objectives described in 5.1, the design is approached in the following way:

1. The LMG simulator is comprised of two real-time tasks running under the RSX-11M operating system on a PDP-11/70 computer. The Command Interpreter task interprets and executes commands given to the LMG simulator. The Shared Memory Display task interprets and displays the contents of shared memory.

2. The high level language FORTRAN IV is used wherever possible.

3. Structured, modular programming techniques are employed for ease of debugging and modification.

4. The simulator is interactive and conversational to facilitate use of the system.

5. An output log file on the disk may be used to record all transactions during a simulation. It can be used for debugging or demonstration purposes.

6. Some commands other than those of the LM and LMG, called 'Maintenance Port' (MP) commands, are defined to support the debugging and demonstration features of the system. The LM, LMG and MP commands supported are detailed in Section 5.3.

7. Two command input modes are offered. Commands can be input either from the CRT or a disk file. Commands and parameters input through the CRT can be recorded to a disk file to be used at a later time as a disk input command file for 'real-time' or repetitive simulations.

8. Indirect command files are used to facilitate the demonstration. The indirect command can contain all the commands and parameters that are necessary for performing the demonstration.

9. The DEC DR11-C General-Purpose Interface Module is used as an interface to transfer data between the PDP-11/70 and the SM of the LM. The DR11-C Driver is written in assembly language and is installed as a loadable driver in the RSX-11M operating system.

5.2.2 STRUCTURE OF THE SIMULATOR

Figures 54 and 55 are system diagrams of the LMG simulator. There are two tasks in the LMG simulator system: Command Interpreter (CI) task and SM Display (SMD) task. The CI task contains four major modules: command identification module, command execution module, local processing module and SM handler module. The command identification module receives and interprets commands. The command execution module includes the Maintenance Port (MP) command routines, LMG command routines and LM command routines.

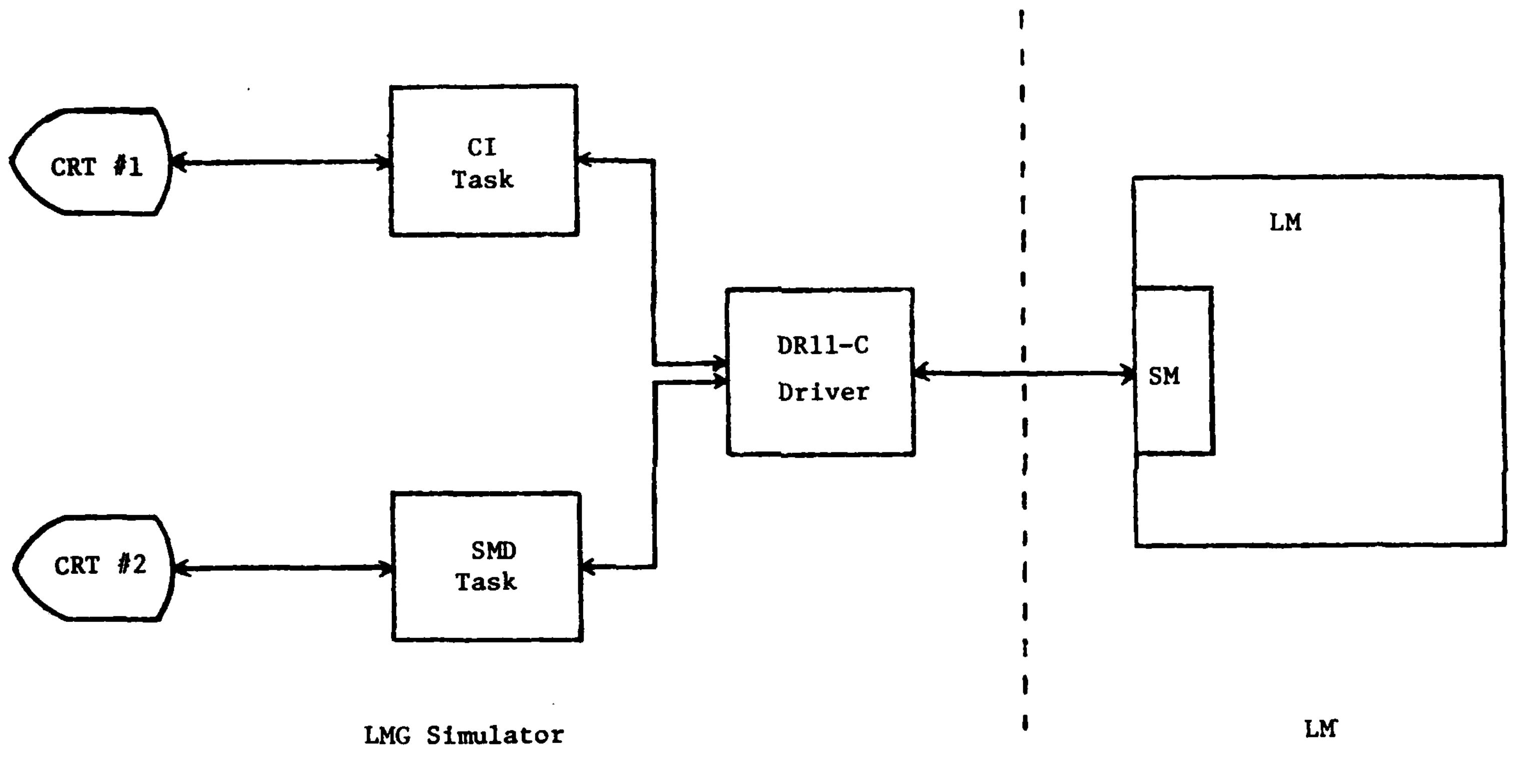


Figure 54 Block Diagram of LMG Simulator

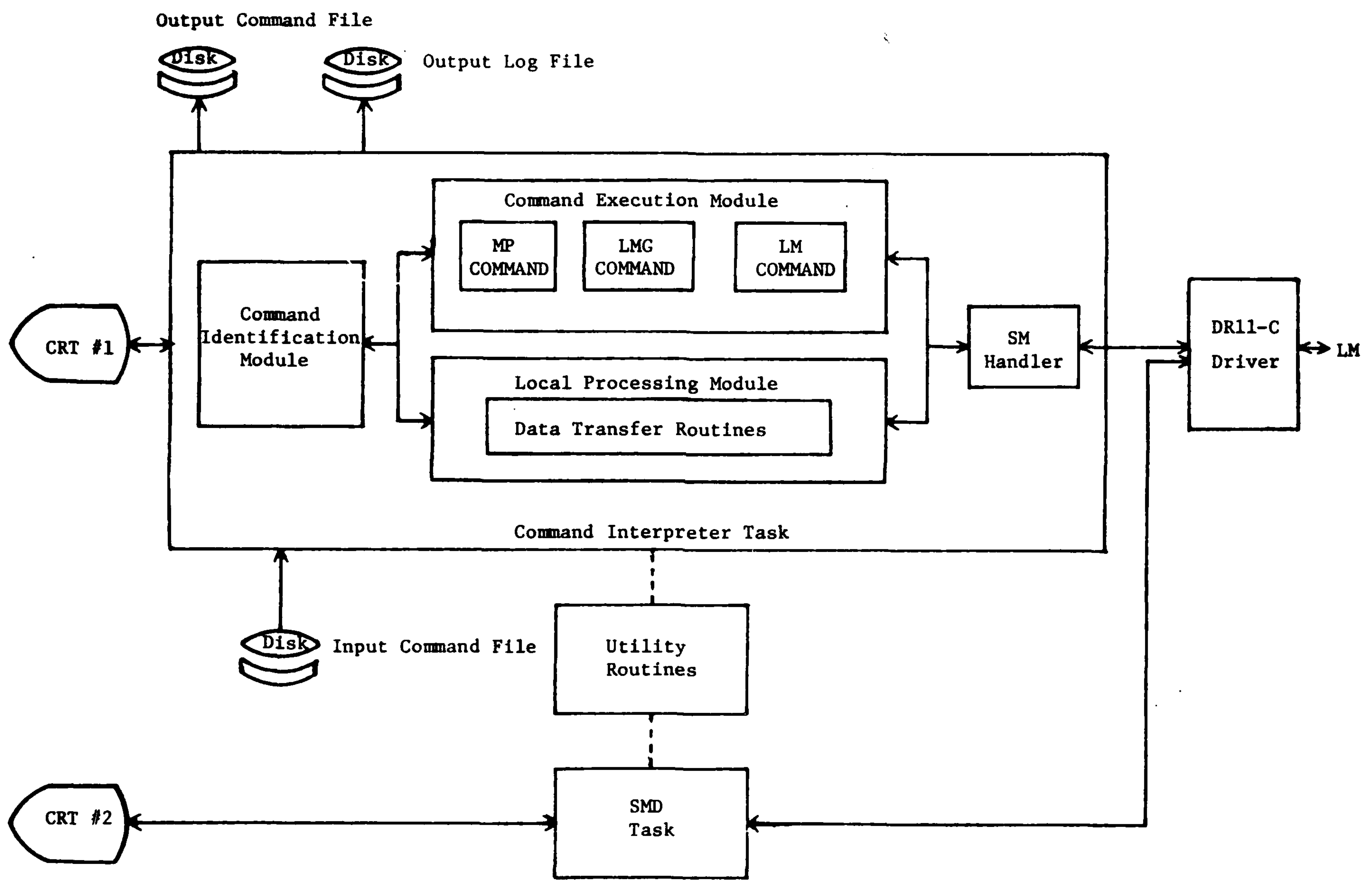


Figure 55 Block Diagram of LMG Simulator System

These routines execute specific functions of the MP, LMG and LM commands. The local processing module includes data transfer routines to perform the data transfer functions between the LMG and the subsystem. The shared memory handler is a subroutine used to access the shared memory through the DR11-C driver. The SMD task uses the DR11-C driver to read the data in the SM, and then interprets it and displays it on the CRT.

The DR11-C driver is an I/O driver written for the RSX-11M operating system to transfer data between the DR11-C and the SM in the LM. Utility Library (UTILIB) is a collection of several utility routines shared by the CI task and SMD task.

5.2.2.1 CI Task

The CI task runs on CRT #1. Commands and parameters can be input from CRT or disk. The command identification module receives a command and parameters, interprets it, and executes it by calling the appropriate command module. It echoes the command and parameters on the CRT if in disk input mode, records the command and parameters to a new command file if in creating command file mode, and records all the transactions during a simulation to a log file if there is one open.

The command module gives command and parameters to, and handshakes with, the SM by calling the SM handler to call the DR11-C driver. Some LMG commands can be executed just by calling a sequence of the LMG modules. The 'LOCAL' command will cause the command identification module to call the LP module to perform data transfer functions to/from the LM using the SM handler and DR11-C driver.

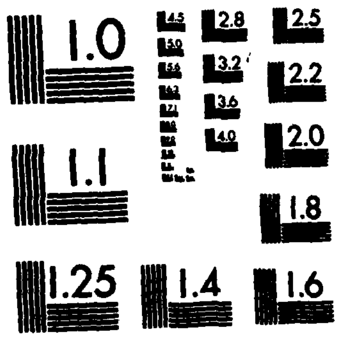
5.2.2.2 SMD Task

The SMD task runs on CRT #2 to continually interpret and display

AD-A124 621 THE REMOTE LINK UNIT: A DEMONSTRATION OF OPERATIONAL PERFORMANCE PART III..(U) HOUSTON UNIV TX DEPT OF ELECTRICAL ENGINEERING C J TAVORA ET AL. AUG 81 3/3

UNCLASSIFIED AFWAL-TR-81-1131-PT-3-1 F33615-80-C-1095 F/G 17/2 NL

END
FILMED
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

the contents of the SM through calls to the DR11-C driver. The display of the SM on CRT #2 is in an easy-to-read, interpreted format. Command name, processing status, special conditions, and all the significant contents of the SM are shown.

5.2.2.3 Data Transfer Routines

The data transfer routines are application programs to perform the data I/O functions. There are four kinds of data I/O: sequential input, sequential output, refreshed input and refreshed output. The data I/O task should be downloaded to the LM first. The LOCAL command causes one of the data transfer routines to communicate with the data I/O task running in the LM to transfer data between the LMG and the subsystem.

5.2.2.4 Utility Routines

The utility routines contain many frequently used subroutines, commonly used by the CI task and the SMD task.

5.2.2.5 SM Handler and DR11-C Driver

The SM handler is a subroutine which has several entry points to access different places in the SM using the DR11-C driver. The DR11-C driver handshakes with the SM hardware to implement data transfer between the LMG simulator (PDP-11) and the LM.

5.3 COMMAND INTERPRETER (CI) TASK

Figure 56 is the structure of CI task. The CI task includes a main program (command identification module) and many subroutines (for example, LMG command routines, LM command routines, MP command routines, LP module, SM handler). It interprets commands and executes them through the appro-

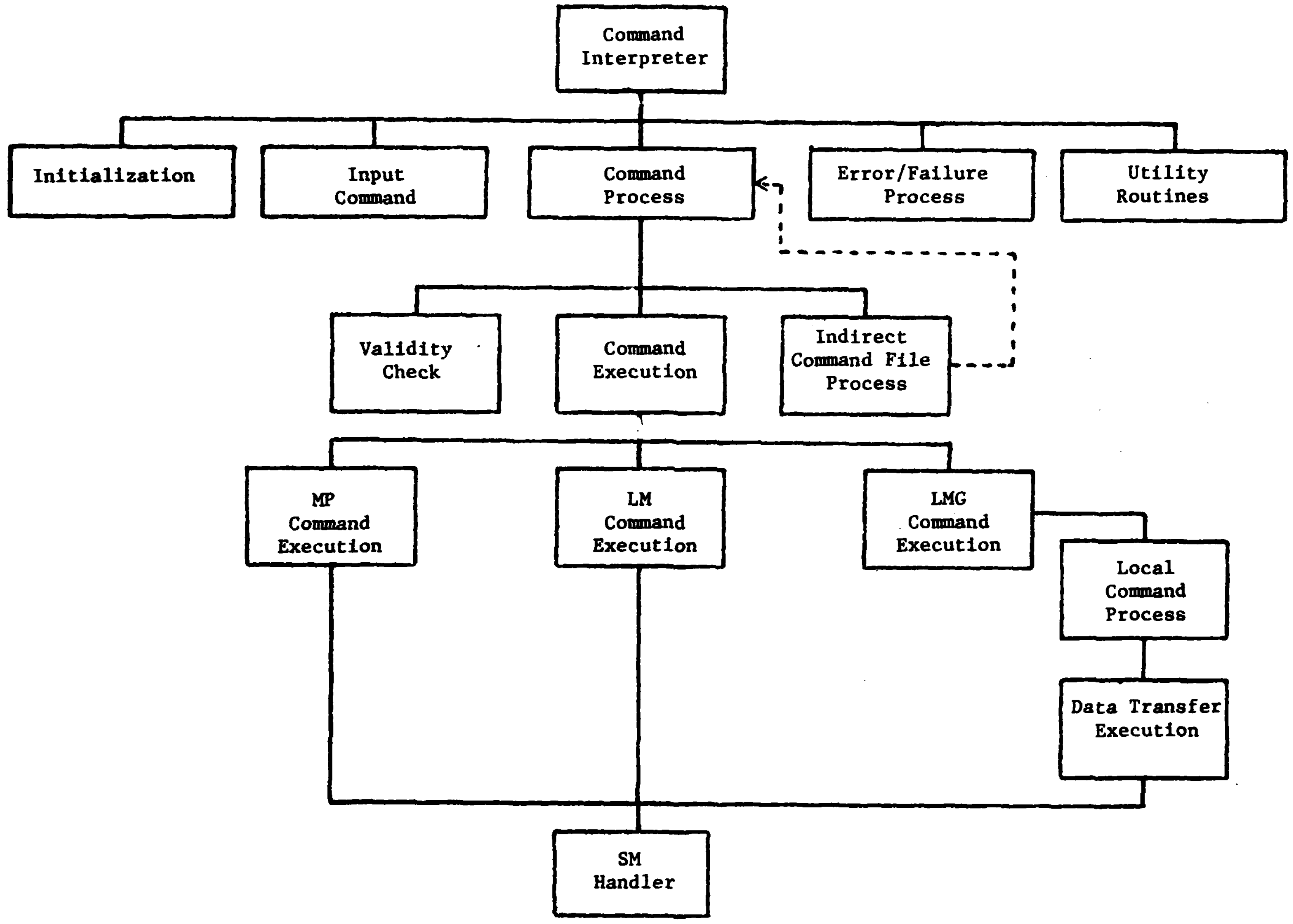


Figure 56 Structure of the Command Interpreter Task

priate modules.

The CI task is initially in CRT input mode, and can be switched between CRT and disk input modes. It can be directed to create a new command file. Also, the action to be taken on errors, whether to stop or continue, can be selected by the user.

The CI task performs initialization first. The initialization steps include requests at CRT #1 for:

1. the cursor home up control characters for the terminal,
2. the type of transcript to the CRT desired for disk input mode,
3. the output log file name, if there is to be one.

The CI task is thenready for a command.

When the CI task receives a command, it first checks the validity of the command. If the command is valid, it is given to the command execution module for execution. If the command is a 'LOCAL' command, a data transfer application subprogram will be invoked to perform the data transfer function between the LMG and the subsystem.

If the input command begins with the character '@', it is an indirect command file rather than a command. The indirect command file procedure opens the indirect command file and processes it using the command process module, as the dashed line indicates in Figure 56. If an error or failure occurs, the CI task processes it to display the error or failure message. The utility routines are used to support the CI task performing its function.

There are three categories of commands: LMG command, LM function command and MP command. The LMG commands are listed in Table 27. The LM function commands are listed in Table 28. The MP commands are listed in

Table 27

LMG COMMANDS

Command	Function
SMDIAG	Perform SM diagnostic function
LOCAL	Request data transfer application program to transfer data between the LMG and the subsystem.

Table 28

LM FUNCTION COMMAND

Command	Function within LM
NOOP	No operation
PRGMLD	Load non-resident task
RUN	Run non-resident task
STOP	Stop non-resident task
CONFIG	Configure selected subsystem
XFRTBL	Transfer LM system tables to/from LMG
CANCEL	Cancel the previous command that is pending
RESET	Reset CMDITR and ICA
RESTRT	Jump to power up restart location
STATUS	Clear input buffer request bit and service request bit in flag mode
NPINIT	Request NP handler to initialize NP's
NPDIAG	Same as NPINIT

Table 29 . All the commands that will access the SM utilize the SM handler. A brief description of each command is given in the following. Full details for using each command are given in the RLUDS User's Manual.

5.3.1 LMG COMMANDS

When the LMG simulator receives a LMG command at CRT #1, it will execute this command by itself, or by issuing a series of LM function commands to the LM, or by a combination of both methods.

1. SMDIAG: The SMDIAG command executes a diagnostic on shared memory, verifying the ability to read from and write to the shared memory. It also checks the hardware interrupt caused in the LM when the LMG writes an LM function command into SM.

2. LOCAL: The LOCAL command causes a data transfer application subprogram to perform data transfer between the LMG and the subsystem. Before issuing the 'LOCAL' command, a data I/O task should be loaded to the LM and run to communicate with the data transfer application subroutine in the LMG.

5.3.2 LM FUNCTION COMMANDS

When the LMG receives an LM function command, it will request the LM to execute this command. The LM will return the status of the command execution and/or the results of execution to the LMG.

1. NOOP: This command doesn't request the LM to take any action. It is used to provide the LMG with a means of exercising the command handshake without causing anything to happen.

Table 29

MP COMMANDS

Command	Function
CRT	Input from CRT
DISK	Input from disk
CLOLF	Close an old log file and open a new one, if wanted.
OPICF	Open input command file
CLICF	Close input command file
OPOCF	Open output command file
CLOCF	Close output command file
CREATE	Creating Output Command File
NOCRET	Stop creating output command file
STPERR	Stop on error
CNTERR	Continue on error
MPSTAT	Output Maintenance Port Status
REWIND	Rewind input command file
EXIT	Exit program
WRITE	Write data to SM
READ	Read data from SM
DUMP	Dump all the data in SM on the CRT
SETTIM	Set PDP-11 system time to LM

2. PRGMLD: This command requests the LM to download a nonresident task from the LMG or upload a nonresident task from the NP. The nonresident task can be a data I/O task or a subsystem diagnostic task.

3. RUN: This command requests the LM to activate the nonresident task in the LM.

4. STOP: This command requests the LM to terminate the running of the nonresident task in the LM.

5. CONFIG: This command requests the LM to cause the ICA handler to configure the ICA with configuration parameters from either the LMG or the NP.

6. XFRTBL: This command requests the LM to transfer LM system tables to or from the LMG. The LMG interprets any table obtained from the LM and displays it on the CRT in an easy to read format.

7. CANCEL: This command requests the LM to stop the execution of any other command already in progress.

8. RESET: This command requests the LM to reset the state of the LM's CMDITR task and of the ICA hardware.

9. RESTRT: This command requests the LM to jump to its power up restart location.

10. STATUS: This command clears the service request bit in the status alert byte and requests the LM to clear the input buffer request bit in flag mode data transfers.

11. NPINIT: This command requests the NP handler to reset all the NPs and cause each NP to run its internal diagnostic. The LMG interprets and displays the diagnostic result on the CRT.

12. NPDIAG: This command is identical to the NPINIT command.

5.3.3 MP COMMANDS

MP commands implement features for debugging and demonstration of the RLU system. They also implement utility functions in the LMG simulator.

1. CRT: This command directs that commands and parameters will be input from the CRT.

2. DISK: This command directs that commands and parameters will be input from the disk (an input command file).

3. CLOLF: This command closes an old output log file and allows the option to open a new output log file.

4. OPICF: This command opens an input command file for use in disk input mode.

5. CLICF: This command closes the opened input command file.

6. OPOCF: This command opens an output command file for recording the command stream (for future use as an input command file).

7. CLOCF: This command closes the opened output command file.

8. CREATE: This command puts the system into the creating mode. All the commands and parameters will be recorded in the output command file.

9. NOCRET: This command puts the system into the non-creating mode.

10. STPERR: This command causes disk input mode execution to stop if an error occurs. At the same time, the disk input mode will be switched to CRT input mode.

11. CNTERR: This command causes disk input mode execution to continue even if an error occurs.

12. MPSTAT: This command will display the maintenance port status on the CRT. The MP status includes the files opened, stop-on-error or continue-on-error condition, creating or non-creating mode, CRT input or disk input mode.

13. REWIND: This command rewinds the input command file to the beginning.

14. EXIT: This command closes all opened files and terminates the CI task.

15. WRITE: This command writes a block of data in HEXASCII format into the SM.

16. READ: This command reads a block of data from the SM and displays it in HEXASCII format on the CRT.

17. DUMP: This command reads all the data in the SM and displays it in HEXASCII format on the CRT.

18. SETTIM: This command gets the time from PDP-11 system and sets the time in the LM.

5.4 SHARED MEMORY DISPLAY (SMD) TASK

The SMD task is a program written to continually interpret and display the contents of the SM on a CRT. Figure 57 is the structure of SMD task.

The SMD task performs initialization first. The initialization steps include requests at CRT #2 for:

1. The cursor home up control characters for the terminal.

2. The refresh rate for updating the SM display on the CRT.

At the user-selected refresh rate the entire contents of the SM will be copied into a buffer in the SMD task using the DR11-C driver. Then the SMD task interprets the data and displays it in HEXASCII on the CRT.

The format of the shared memory display is shown in Figure 58. The information displayed includes:

1. The most recent LM function command name and its processing status.

2. Information on special conditions such as a service request, LM alert, NP alert, ICA alert, LA alert and subsystem down.

3. Command and status bytes in HEXASCII format. The command bytes include the LM function command and the data transfer command. The status byte include LM function status, data transfer status, status alert, LM status, NP status, ICA status, and LA status.

4. Contents of the LM buffer, I/O buffer 1 and I/O buffer 0 in HEXASCII.

5.5 DATA TRANSFER ROUTINES

The data transfer routines are application routines to transfer data

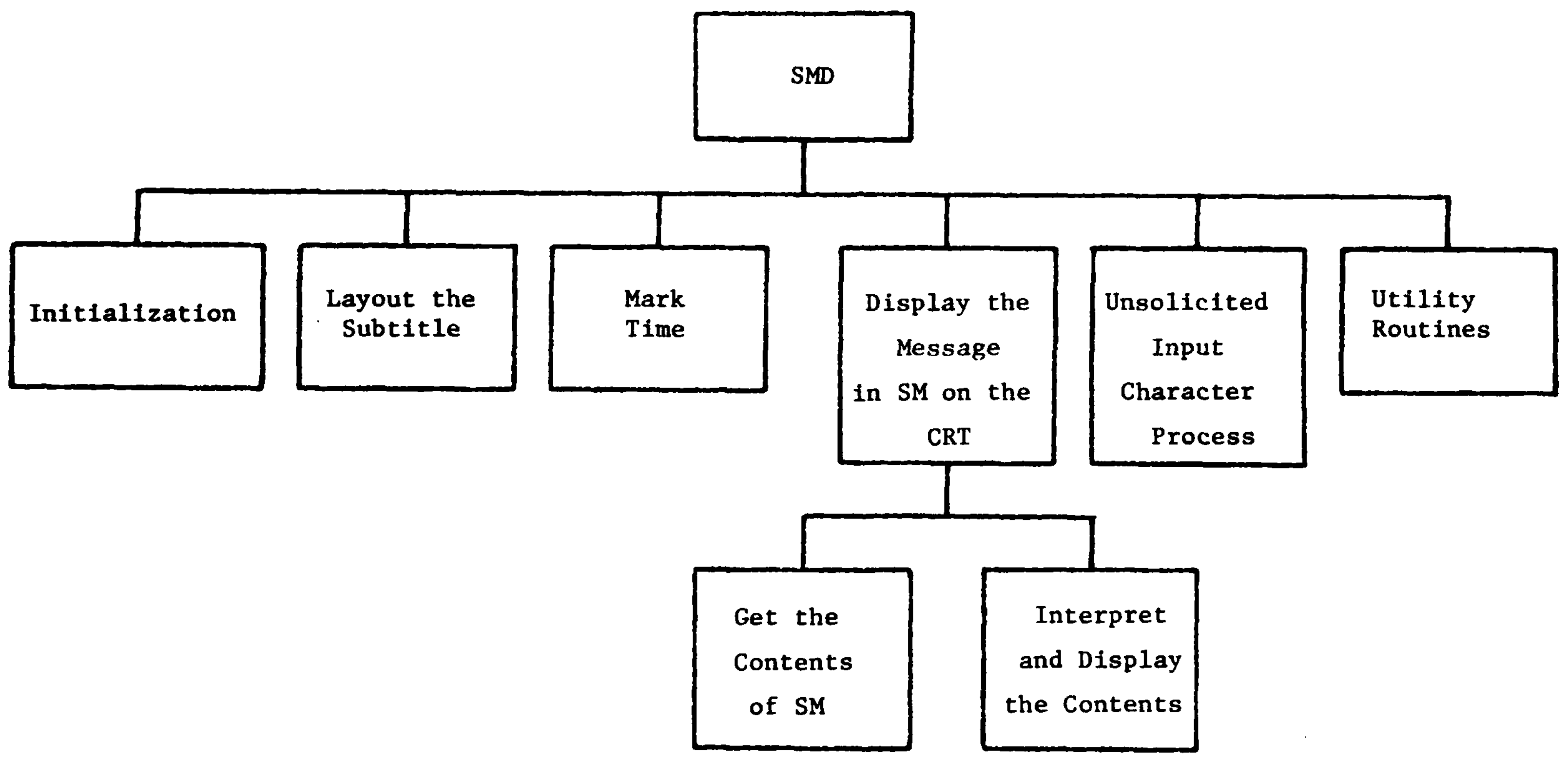
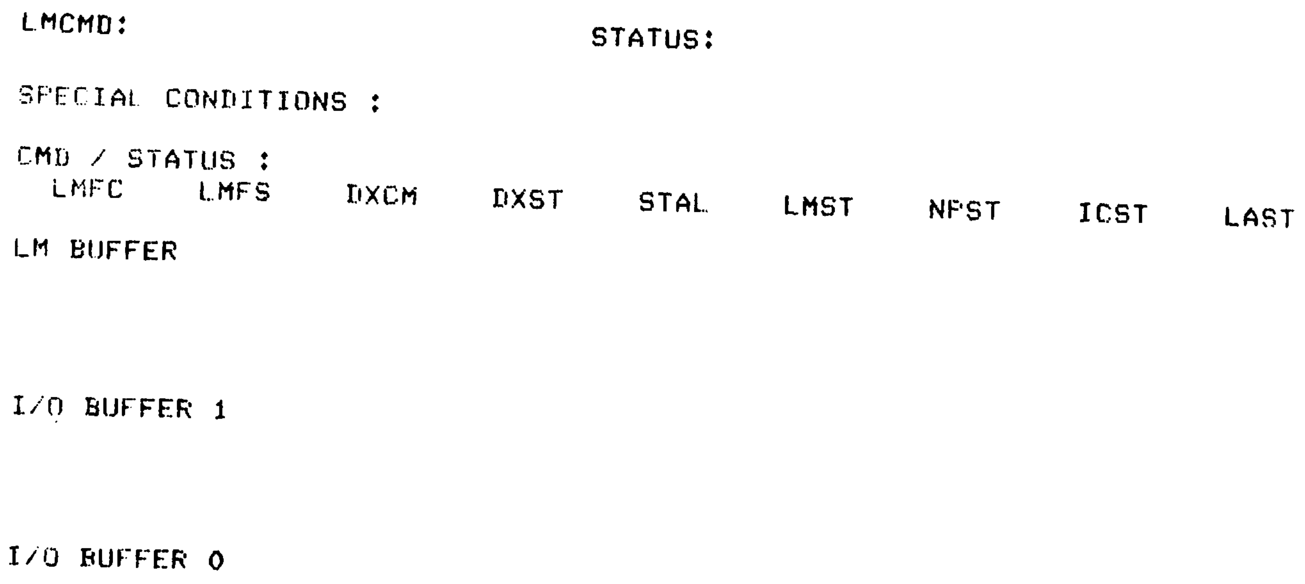

SMD
Initialization
Layout the Subtitle
Mark Time
Display the Message in SM on the CRT
Unsolicited Input Character Process
Utility Routines
Get the Contents of SM
Interpret and Display the Contents

Figure 57 Structure of Shared Memory Display Task



```
LMCMD:                                STATUS:

SPECIAL CONDITIONS :

CMD / STATUS :
  LMFC     LMFS     DXCM     DXST     STAL     LMST     NFST     ICST     LAST
LM BUFFER



I/O BUFFER 1



I/O BUFFER 0
```

Figure 58 **Layout of Shared Memory Display**

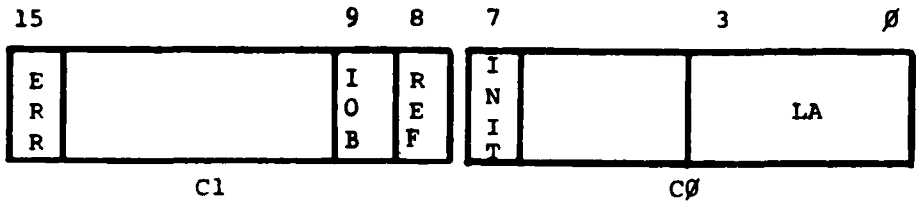
between the LMG and the subsystem. A data I/O nonresident task should be loaded into the LM from either the LMG or the NP. The nonresident task handshakes through SM with the data transfer routines to perform data transfer to or from LMG. There are four types of data transfers: sequential input, sequential output, refreshed input and refreshed output. Figure 59 shows the protocol of the data transfer handshake. Figure 60 shows the structure of the data transfer modules. The details of each are given below.

5.5.1 DATA TRANSFER HANDSHAKE PROTOCOL

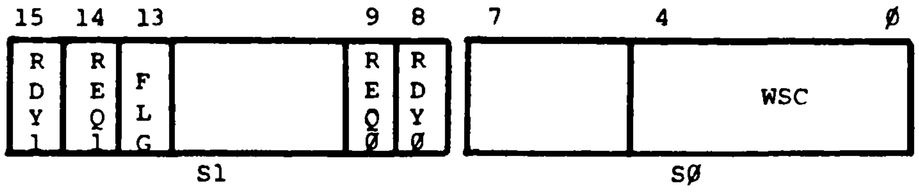
Figure 59 shows the command byte and status bytes used in the data transfer handshake. C1 is the data transfer command byte. S1 and SØ are the data transfer status bytes.

The LMG begins a data transfer operation by performing a read-modify-write on status by S1. After the read portion, the LMG checks the RDYØ and RDY1 bits to see if Buffer Ø (output buffer) is available or Buffer 1 (input buffer) is available. If the desired buffer is available, the LMG sets REQØ or REQ1 to indicate which is being taken and writes S1 back to SM. It then writes SØ to SM to give the number of bytes (WSC) to be transferred. After the data is transferred, the LMG writes command byte C1 to SM, causing an interrupt. In C1, IOB indicates whether Buffer 1 or Ø is being released, and REF=1 if buffer being released is refreshed.

Data Transfer Command Word:



Data Transfer Status Word:



Data Transfer Handshake:

1. LMG sends CØ to LM, generating an interrupt

   LA = 4-bit Link Address
   INIT = 1 to indicate initiation of message transfer

2. LMG performs read-modify-write on S1

   FLG = 1 if subsystem is flagged as down
   RDYØ = 1 if buffer Ø is available
   RDY1 = 1 if buffer 1 is available
   REQØ = 1 to indicate buffer Ø is being taken
   REQ1 = 1 to indicate buffer 1 is being taken

3. LMG writes SØ to LM (output) or reads SØ from LM (input)

   WSC = word subcount, no. of words being transferred

4. LMG transfers data

5. LMG sends C1 to LM, generating an interrupt

   ERR = 1 if an error was encountered
   IOB = 1 to release buffer 1, Ø to release buffer Ø (single-buffered I/O)
   REF = 1 if buffer to be released is refreshed

Figure 59 Data Transfer Handshake Protocol

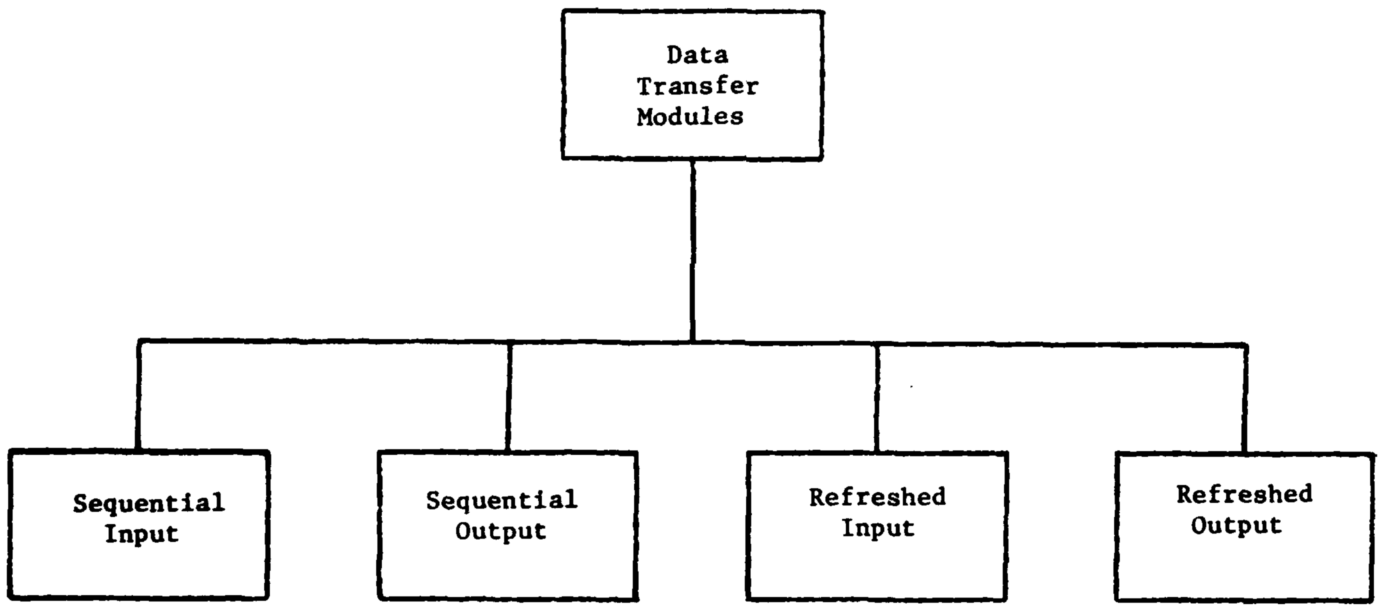


Figure 60   Structure of the Data Transfer Modules

5.5.2 DATA TRANSFER MODULES

Following is the handshake sequence for each of the data transfer modules.

5.5.2.1 Sequential Input

1. LMG requests input buffer (if RDY1=1 and FLG=Ø, then LMG sets RDY1=Ø and REQ1=1).

2. LMG reads the WSC (word subcount) and data from SM.

3. LMG issues STATUS function command to release buffer and clear asynchronous service request.

5.5.2.2 Sequential Output

1. LMG requests output buffer (if RDYØ=1 and FLG=Ø, then LMG sets RDYØ=Ø and REQØ=1).

2. LMG writes WSC and data into SM.

3. LMG writes data transfer command C1 ($ØØ_{16}$), causing an interrupt to the LM to release buffer Ø.

5.5.2.3 Refreshed Input

1. LMG requests input buffer (if RDY1=1 and FLG=Ø, then LMG sets RDY1=Ø and REQ1=1).

2. LMG reads WSC and data from SM.

3. LMG writes data transfer command C1 ($Ø3_{16}$), causing an interrupt to the LM to release the buffer.

5.5.2.4 Refreshed Output

1. LMG requests output buffer (if RDYØ=1 and FLG=Ø, then LMG sets RDYØ=Ø and REQØ=1).

2. LMG writes WSC and data into SM.

3. LMG writes data transfer command C1 ($Ø1_{16}$), causing an interrupt to the LM to release the buffer.

5.6 UTILITY ROUTINES

The utility routines are stored in a file called UTILIB. These routines are frequently used by or shared by other programs or subroutines. They can be classified into general categories by their functions. These general categories are described below.

5.6.1 DATA CONVERSION ROUTINES

These routines perform general purpose data conversion tasks:

1. Conversion between binary data and HEXASCII characters.

2. Conversion between BCD data and BCD ASCII characters.

3. Conversion between decimal values and 2's complement binary or offset binary values.

4. Conversion of binary data to BCD ASCII characters.

5.6.2 DATA TRANSFER ROUTINES

These perform the following handshake protocol functions:

1. Request I/O buffer.

2. Read or write WSC.

3. Read or write data.

4. Write C1 to interrupt LM.

5.6.3 MISCELLANEOUS UTILITY ROUTINES

Some of the major functions performed by this last category include:

1. Wait and check the status of LM function command execution.

2. Clear service request bit.

3. Clear the screen of the terminal.

4. Check one bit of a byte.

5. Read a parameter and check whether its value falls in the predefined range.

6. Logical function 'YES' to facilitate answering yes-or-no questions.

5.7 SM HANDLER

The Shared Memory Handler (SMH) is a subroutine which provides many entry points for other programs to call to access the SM. The SM has 256 bytes, with addresses from ∅ to 255. Each SMH entry point implies a starting address for accessing SM through the DR11-C driver. The number of bytes to be accessed and the function to be performed (read, write or read-modify-write) are passed as arguments in the call to the entry point. Using symbolic entry points for each type of SM access relieves the calling program of the responsibility for knowing any SM addresses or handshake protocols.

5.7.1 SM HANDLER FUNCTIONS

Described below are the functions of each of the entry points of the SMH listed in Table 30.

WFCMD: This entry point is used to write the LM function command byte to the SM.

RFCMD: This entry point is used to read the LM function command byte from the SM.

RFSTS: This entry point is used to read the LM function status byte from the SM.

WXCMD: This entry point is used to write the data transfer command byte (C1) to the SM.

RXCMD: This entry point is used to read the data transfer command byte from the SM.

RS1: This entry point is used to read the data transfer status byte S1 from the SM.

WS1: This entry point is used to write the data transfer status byte S1 to the SM.

RWMS1: This entry point is used to do read-modify-write on data transfer status byte S1.

Table 30

FUNCTIONS OF SMH

Entry Name	Function
WFCMD	Write LM function command byte
RFCMD	Read LM function command byte
RFSTS	Read LM function status byte
WXCMD	Write data transfer command byte
RXCMD	Read data transfer command byte
RS1	Read data transfer status byte S1
WS1	Write data transfer status byte S1
RWMS1	Read-modify-write data transfer status byte S1
WTXSØ	Write data transfer status byte SØ
RTXSØ	Read data transfer status byte SØ
WFBUF	Write data to LM function buffer
RFBUF	Read data from LM function buffer
WXBUF	Write data to data I/O buffer
RXBUF	Read data from data I/O buffer
RLASTS	Read LA status byte
RICSTS	Read ICA status byte
RLMSTS	Read LM hardware status byte
RNPSTS	Read NP hardware status byte
RSTSAL	Read status alert byte
WSTSAL	Write status alert byte

WTXSØ: This entry point is used to write the data transfer status byte SØ to the SM.

RTXSØ: This entry point is used to read the data transfer status byte SØ from the SM.

WFBUF: This entry point is used to write data to LM function buffer. The number of data bytes is limited to 64.

RFBUF: This entry point is used to read data from the LM function buffer. The number of data bytes is limited to 64.

WXBUF: This entry point is used to write data to an I/O buffer. Buffer Ø or buffer 1 can be selected. The number of data bytes is limited to 64.

RXBUF: This entry point is used to read data from an I/O buffer. Buffer Ø or buffer 1 can be selected. The number of data bytes is limited to 64.

RLASTS: This entry point is used to read the LA status byte from the SM.

RICSTS: This entry point is used to read the ICA status byte from the SM.

RLMSTS: This entry point is used to read the LM hardware status byte from the SM.

RNPSTS: This entry point is used to read the NP hardware status byte from the SM.

RSTSAL: This entry point is used to read the status alert byte from the SM.

WSTSAL: This entry point is used to write the status alert byte to the SM.

5.8 DR11-C DRIVER

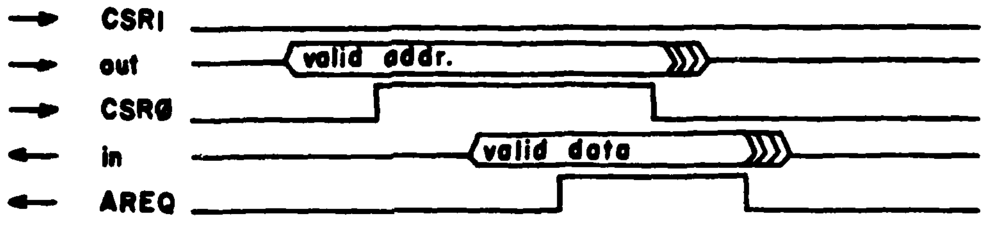
The DR11-C driver is an RSX-11M I/O peripheral driver written in assembly language. It drives the DEC DR11-C board which interfaces the LM with the DEC PDP 11/70 that simulates the LMG.

The DR11-C board has two output control lines labled CSRØ and CSR1 and two input control lines labled AREQ and BREQ. The BREQ line is unused. The high order byte of the 16 data lines is used as the SM address. The low order byte of the 16 data lines is used as the data for input or output.

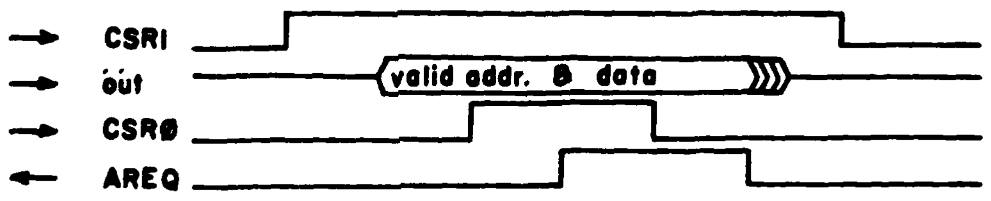
Figure 61 is a diagram of the handshakes for read, write and read-modify-write operations. For the 'write' function, the driver sets CSR1 and puts valid address and data in the output register. Then the driver sets CSRØ to initialize the data transfer. AREQ should be low before setting the CSRØ. The driver then exits and is recalled when AREQ goes high generating an interrupt. The driver then resets CSRØ and waits in a wait loop for AREQ to return low. Timeout checks are wherever needed. The 'read' and 'read-modify-write' functions are similar to 'write', as indicated in Figure 61. The flowchart for the driver is given in Appendix C, Section 5-A.

5.9 DEMONSTRATION EXAMPLE

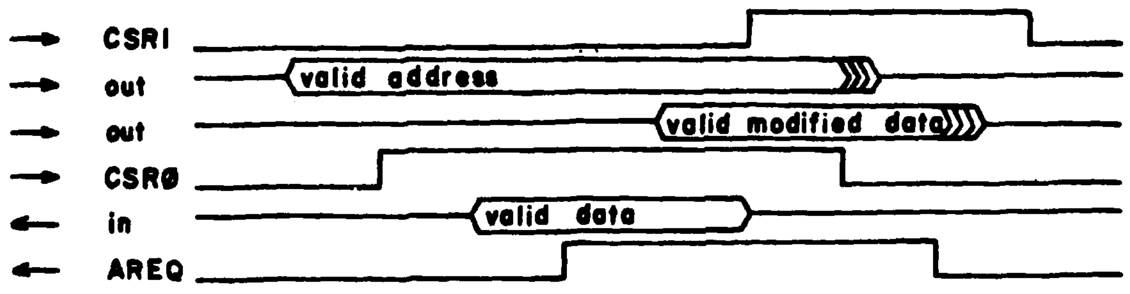
A serial RLU test example is given in this section to illustrate the



(a)



(b)



(c)

Figure 61 Handshake between the LMG and LM for (a) read, (b) write and (c) read-modify-write.

use of the simulator. Figure 62, 63 and 64 are indirect command files named R2A.DEM, R2B.DEM and R2C.DEM, respectively. Figure 65 is a transcript of CRT #1 during this example. Figure 66 shows the SM display on the CRT #2 at three points in the example. Those characters with underlines are input from the CRT.

In Figure 65 (a), the CI task does initialization before prompting for a command. The first command @R2A leading with character @ indicates that R2A is an indirect command file. The indirect command file R2A as shown in Figure 62 is opened and the command stream in this file is executed. Each line in the indirect command file is a command or parameter. (If a line begins with the character ; or !, the CI task treats it as a comment and ignores it.) Because the subsystem is not attached to the LM at first, the return status from the NPINIT command is failure code -1∅. At the same time, referring to Figure 66 (a), the'NP ALERT' message is shown on the CRT #2 screen as a special condition to indicate that no subsystem is connected to the LM. The LM function command NPINIT and return status failure code -1∅ are also displayed on the CRT. The transcript is displayed on the CRT #1 because of the selection of 'full transcript on CRT' during the initialization. If 'brief transcript on CRT' is chosen, only the commands and parameters will be displayed on CRT. If 'no transcript on CRT' is chosen, nothing will be displayed on CRT except a return status of 'failure'.

Now we connect the serial subsystem to the LM and issue the @R2A command again. All the procedures performing the serial subsystem test can be seen on the CRT #1 as shown in Figure 65b. The NPINIT initializes and diagnoses the NP. The diagnostic result shows that the NP has 8 free re-

```
;TEST: SERIAL INPUT/OUTPUT (UP LOAD FROM NP)
;       INPUT: GROUP B  ;  OUTPUT: GROUP A
;
;INITIALIZE NP
;
NPINIT
;
;LOAD DIO TASK FROM NP
;
PRGMLD
DIO
NP
;
;CONFIGURE GROUP A AS SERIAL OUTPUT FROM NP
;
CONFIG
NP
A
;
;CONFIGURE GROUP B AS SERIAL INPUT (FLAG) FROM NP
;
CONFIG
NP
B
;
;RUN NON-RESIDENT TASK
;
RUN
;
;ASK FOR LOCAL PROCESSING TASK - SERIAL INPUT/OUTPUT
;
LOCAL
SIO
;
;STOP NON-RESIDENT TASK
;
STOP
```

Figure 62 Indirect Command File R2A.DEM

```
XFRTBL
NPREC
W
3
1
XFRTBL
NPREC
R
1
```

Figure 63 Indirect Command File R2B.DEM

```
;RUN NON-RESIDENT TASK
;
RUN
;
;ASK FOR LOCAL PROCESSING TASK - SERIAL INPUT/OUTPUT
;
LOCAL
SIO
;
;STOP NON-RESIDENT TASK AND RESET LM
;
STOP
RESET
```

Figure 64 Indirect Command File R2C.DEM

```
LOGI
RUN CMDINI

CHECK AND ANSWER THE QUESTIONS ABOUT
THE CURSOR HOME UP CONTROL FOR YOUR TERMINAL
   EXAMPLE OF CONTROL CHARS FOR CURSOR HOME UP :
       ADM3A         1 CHAR     --- 1E
       ACT 4         1 CHAR     --- 1D
       DEC VT-52     2 CHARS    --- 1B,48
       HAZE 1500     2 CHARS    --- 7E,12
       HARD COPY     1 CHAR     --- 00

# OF CONTROL CHARS ? 2
ENTER CONTROL CHARS IN HEXASCII AS XX,XX
1B,48

SELECT ONE OF THE FOLLOWING :
 0 : NO TRANSCRIPT ON CRT
 1 : BRIEF TRANSCRIPT ON CRT
 2 : FULL TRANSCRIPT ON CRT
2

DO YOU WANT TO OPEN A LOG FILE ?(Y/N)Y
OPEN OUTPUT LOG FILE
ENTER FILENAME
SERIALTST
$$$$ OUTPUT LOG FILE "SERIALTST.LOG" IS OPENED $$$$
ENTER IDENTIFICATION
TEST SERIAL SUBSYSTEM
***************************************************************************

    REMOTE LINK UNIT DEMOSTRATION       (sic)
         17:15:40          09-JAN-81
ID: TEST SERIAL SUBSYSTEM

***************************************************************************

CMD> @R2A
NPINIT
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER NPINIT CMD MODULE ******


STATUS: FAILURE  CODE: -10
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 65 (a) Display on CRT #1

```
CMD> @R2A
NPINIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER NPINIT CMD MODULE ******

NP DIAGNOSTIC RESULT :
    ** DIAGNOSTIC COMPLETED
    # OF FREE RECORDS =   8

STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PRGMLD
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER PRGMLD CMD MODULE ******

DATA I/O (DIO) OR SUBSYSTEM DIAGNOSTIC (SSD) ?
DIO
FROM LMG OR NP ?
NP

STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CONFIG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER CONFIG CMD MODULE ******

FROM LMG OR NP ?
NP
GROUP A OR B ?
A

STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CONFIG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER CONFIG CMD MODULE ******

FROM LMG OR NP ?
NP
GROUP A OR B ?
B

STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
RUN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER RUN CMD MODULE ******


STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LOCAL
```

Figure 65 **(b) Display on CRT #1**

```
SELECT ONE OF THE DATA TRANSFER ROUTINES :
REFIN     ALGIN     SYNIN     SINREF    SAMDIN    DINMOF
SEQIN     SINFLG    DINMOL
SEQOUT    ALGOUT    SYNOUT    DISOUT    SOUTA     SOUTB
SYNIO     SIO       CRTSIM
?
SIO




     00 + 00 = 0000

     09 * 04 = 0036

     09 + 04 = 0013


< SUBSYSTEM DOWN >
STOP
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER STOP CMD MODULE ******


STATUS: FAILURE  CODE: -11
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 65 **(b) (Continued)**

```
CMDD @R2B
XFRTBL
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER XFRTBL CMD MODULE ******

TABLE NAME ?
NPREC
READ OR WRITE (R/W) ?
W
RECORD FUNCTION (0 - 5)
0 : WRTREC       1 : EOD
2 : BOD          3 : BACREC
4 : FWDREC       5 : ERAWRT
?
  3
# OF RECORDS (1 - 16)
  1

STATUS: SUCCESS
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
XFRTBL
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER XFRTBL CMD MODULE ******

TABLE NAME ?
NPREC
READ OR WRITE (R/W) ?
R
# OF RECORDS (1 -  3)
  1

3A  07  09  00  18  05  07  FF  FF  FF  FF  FF  FF  FF  99  FF

RECORD ID CODE : 3A
    SUBSYSTEM FAILURE DETECTED
    # OF VALID BYTES :    7
    DATE: JAN    9 ( JAN    9 FOR LEAP YEAR)
    TIME:   18 :   5 (HOUR:MINUTE)
```

Figure 65   **(c) Display on CRT #1**

```
CMD> @R2C
RUN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
****** ENTER RUN CMD MODULE ******


STATUS: SUCCESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
LOCAL

SELECT ONE OF THE DATA TRANSFER ROUTINES :
REFIN     ALGIN     SYNIN     SINREF    SAMDIN    DINMOF
SEQIN     SINFLG    DINMOL
SEQOUT    ALGOUT    SYNOUT    DISOUT    SOUTA     SOUTB
SYNIO     SIO       CRTSIM
>
SIO




  00 + 00 = 0000

  09 * 04 = 0036

  STOP
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  ****** ENTER STOP CMD MODULE ******


  STATUS: SUCCESS
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  RESET
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  ****** ENTER RESET CMD MODULE ******


  STATUS: SUCCESS
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  CMD> E
  ************************************************************************

         END OF DEMONSTRATION

  ************************************************************************
  TT7  --  STOP
```

Figure 65 **(d) Display on CRT #1**

```
IOSB=***1
LMCMD: NPINIT                        STATUS: FAILURE      -10
SPECIAL CONDITIONS :

     NP ALERT
CMD / STATUS :
  LMFC     LMFS      DXCM      DXST      STAL      LMST      NPST      ICST      LAST
   0B       F6        00       0000       00       0000      0000      0000       00
LM BUFFER
      01   02   01   13   00   00   00   00   00   18   07   FF   FF   FF   FF   FF
      FF   FF   87   FF   FC   FC   FC   DC   FC   FF   B4   DC   7C   FC   FC   FC
      74   FC   F8   7C   FC   FD   EC   FC   FC   FD   F8   BC   74   FD   FC   FC
      7C   BC   F4   FC   FC   FC   98   FC   76   FD   DC   FC   FC   FC   FC   FC
I/O BUFFER 1
      00   00   00   00   00   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   8D   00   FF   EF
I/O BUFFER 0
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   C9   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
      76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
```

Figure 66 (a) SM Display on CRT #2

```
IOSB=***1
LMCMD: STATUS                      STATUS: SUCCESS
SPECIAL CONDITIONS :


CMD / STATUS :
  LMFC     LMFS     DXCM     DXST     STAL     LMST     NPST     IOST     LAST
   08       00       00      0105      00      0000     0030     0400      02
LM BUFFER
     01  02  01  00  00  00  00  00  00  18  07  FF  FF  FF  FF  FF
     FF  FF  87  FF  FC  FC  FC  DC  FC  FF  B4  DC  7C  FC  FC  FC
     74  FC  F8  7C  FC  FD  EC  FC  FC  FD  F8  BC  74  FD  FC  FC
     7C  BC  F4  FC  FC  FC  98  FC  76  FD  DC  FC  FC  FC  FC  FC
I/O BUFFER 1
     09  10  04  36  00  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  8D  00  FF  EF
I/O BUFFER 0
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  C9  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  89  00  FF  EF
     76  FF  00  10  89  00  FF  EF  76  FF  00  10  89  00  FF  EF
```

Figure 66 (b) SM Display on CRT #2

```
IORB=***1
SMCMD: STOP                        STATUS: FAILURE      -11
SPECIAL CONDITIONS :

    SUB DOWN
CMD / STATUS :
  LMFC     LMFS     DXCM     DXST     STAL     LMST     NPST     IOST     LAST
   02       F5       00      2005      00      0000     0030     FD00      B7
LM BUFFER
        01   02   01   00   00   00   00   00   00   18   07   FF   FF   FF   FF   FF
        FF   FF   87   FF   FC   FC   FC   DC   FC   FF   B4   DC   7C   FC   FC   FC
        74   FC   F8   7C   FC   FD   EC   FC   FC   FD   F8   BC   74   FD   FC   FC
        7C   BC   F4   FC   FC   FC   98   FC   76   FD   DC   FC   FC   FC   FC   FC
I/O BUFFER 1
        09   00   04   13   00   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   8D   00   FF   EF
I/O BUFFER 0
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   C9   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
        76   FF   00   10   89   00   FF   EF   76   FF   00   10   89   00   FF   EF
```

Figure 66 (c) SM Display on CRT #2

cords. The Data I/O task is uploaded from the NP. Group A and Group B of the ICA are both configured from the NP. The nonresident data I/O task is executed by the RUN command. The LOCAL command offers a choice of data transfer routines. In this example, the serial I/O (SIO) data transfer routine is selected. Two numbers, an operator, and the result are transferred to the LMG and displayed on the CRT when the subsystem flag is raised. For example, the first data transfer is displayed on CRT #1 as 09*04=0036. At the same time, referring to Figure 66 (b), we can see these data in the input buffer (I/O Buffer 1)

Now we change the transmission parity of the subsystem. A 'subsystem down' message is shown on CRT #1 and the data I/O task stops running. Therefore, the STOP command gets return status failure code -11. This failure code means that the nonresident task had already stopped running. This same condition can be seen at CRT #2 as shown in Figure 66 (c).

In Figure 65 (c), @R2B is issued to examine the time and cause of the subsystem failure.

In Figure 65 (d), @R2C is issued after correcting the errors that caused the subsystem failure. The Data I/O task restarts and the local processing routine performs the data transfers. At the end of the demonstration, we stop the data I/O task and reset the ICA configuration. The E(EXIT) command is used to end the demonstration.

SECTION 6

SUBSYSTEMS

Subsystems are peripheral components that can be interfaced to an avionic system through an LM. Two subsystems have been designed for the RLU demonstration: a serial subsystem and a synchro subsystem. The former sends and receives serial data, while the latter handles analog data in the form of synchro voltages. Attached to each subsystem is a nameplate which holds data pertaining to its subsystem. When the LM is connected to a subsystem, it must be appropriately configured to be able to successfully transfer data. The ICA configuration required for a subsystem is stored in the subsystem's nameplate.

Each subsystem has programs stored in its nameplate which will run in the LM when the subsystem is connected. Each program enables data to be input from the subsystem, performs certain operations on the data, and then outputs the processed data to the subsystem.

Discussion of the design of each subsystem requires that three important aspects be considered: the hardware of the subsystem, the related software necessary, and the data stored in the corresponding nameplate. This section will discuss these three aspects for both the serial and synchro subsystems.

6.1 SERIAL SUBSYSTEM

This subsystem is capable of transmitting three 8 bit bytes as serial data to the LM. It can also receive and display two 8 bit bytes from the LM as serial data. The data to be transmitted can be set through thumbwheel switches. The subsystem generates a parity bit for every byte of data to be transmitted and is capable of performing a parity check on data received. The data transfers are synchronous with the clock being provided by the LM.

Demonstration of the serial subsystem takes place by using the subsystem as a calculator that performs additions and multiplications on double-BCD-digit operands. Of the three bytes transmitted by the subsystem, the first and third bytes define the double-BCD-digit operands, while the second byte defines the operation. The LM performs the operation and produces a four-BCD-digit result. This result (two bytes) is then transmitted by the LM and received and displayed by the subsystem.

6.1.1 HARDWARE DESIGN

The subsystem consists of two sections: the sending subsystem, and the receiving subsystem. The sending subsystem can operate in two modes - flag or refresh. In the flag mode data is transmitted only when the STBSW switch on the panel is closed. Prior to closure of STBSW the value of the thumbwheel switches can be set. On closure of STBSW, data on the thumbwheel switches is latched until either the transmission is successful, or the RESET switch on the panel is closed. In the refresh mode data is transmitted as soon as it is requested by the LM. Data is latched only as long as the transmission takes places. Changes on the thumbwheel switches are detected during the periodic transmission cycles initiated by the LM.

The block diagram of the sending subsystem is shown in Figure 67. A detailed drawing of this circuit is presented in Appendix B, Section 6-A. block diagram there are two inputs REQ/LOK and SCLK from the ICA and two outputs SDATA and FLG/ACK to the ICA. The ICA requests transmission by

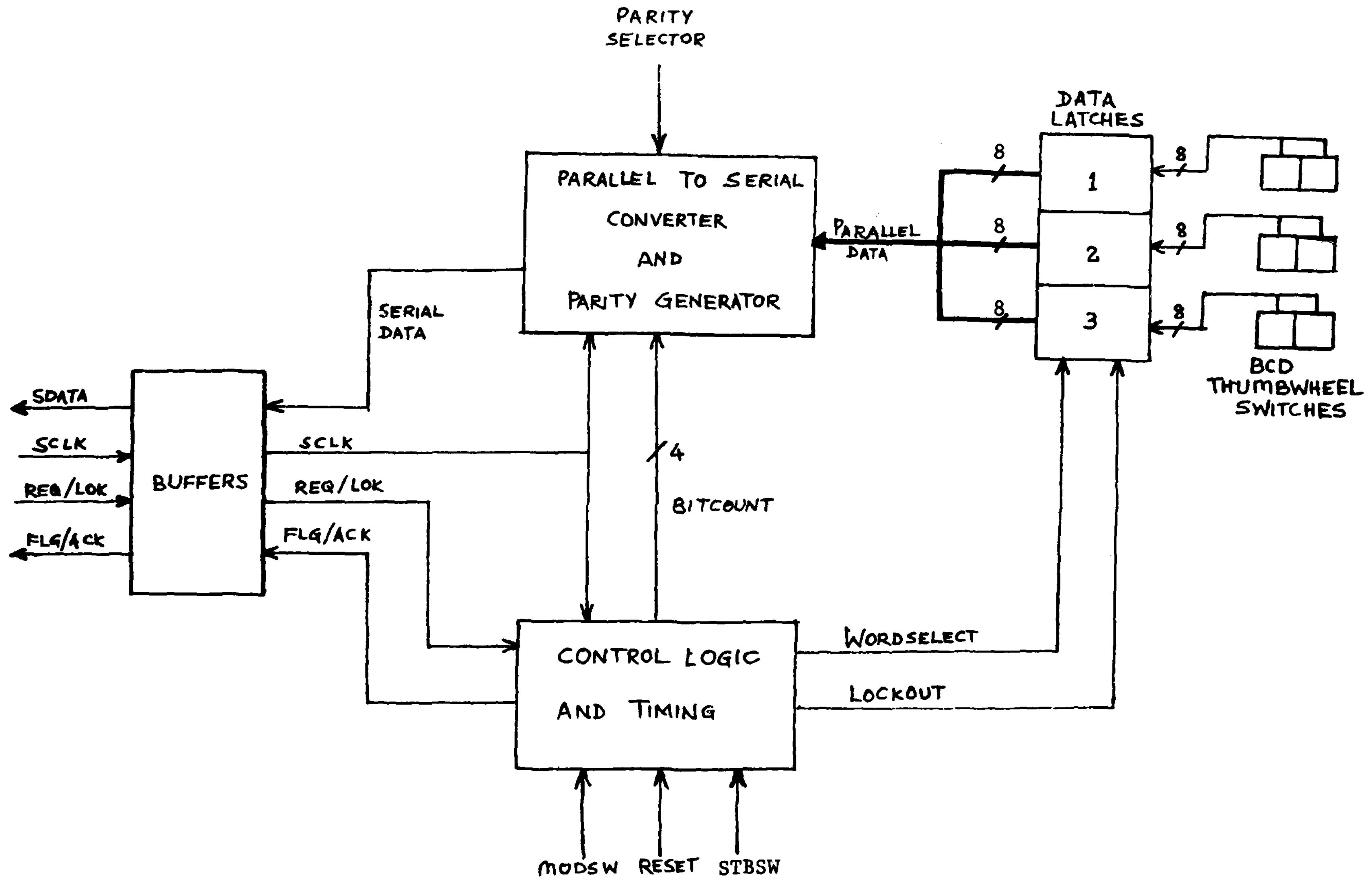


Figure 67 Block Diagram of Serial Sending Subsystem

raising the REQ/LOK line. If MODSW is in the refresh mode the control logic immediately responds by raising the FLG/ACK line. If MODSW is in the flag mode, the control logic will raise FLG/ACK only when STBSW is closed. When FLG/ACK goes high the data on the thumbwheel switches is latched into the data latches by LOCKOUT.

When the ICA sees FLG/ACK high, it sends the clock pulses on SCLK. The control logic uses SCLK to generate BITCOUNT and 'WORDSELECT'. Depending on 'WORDSELECT' an 8 bit byte is selected from the outputs of one of the three latches. 'BITCOUNT' determines which bit of this byte is to be transmitted. That bit is output from the parallel to serial converter. When the bit count goes to 9 the parity generator transmits the parity bit in accordance with the PARITY selector switch. When 3 bytes have been transmitted either of two things may happen depending on the mode. In the refresh mode the last clock pulse goes low and then REQ/LOK goes low. This sets LOCKOUT low and the data is no longer latched. On the other hand, in the flag mode LOCKOUT needs to be high if the transmission was unsuccessful. To keep LOCKOUT high the ICA brings REQ/LOK low first, and then brings the last clock pulse low. This keeps the old data latched. If the transmission was successful in the flag mode then the ICA brings the last clock pulse low and then brings REQ/LOK low (similar to the refresh mode). LOCKOUT can be set low also by closing RESET.

The second section of the hardware is the receiving subsystem. The block diagram for the receiving subsystem is shown in Figure 68. A detailed drawing of this circuit is presented in Appendix B, Section 6-A. As shown in the block diagram, there are 3 input lines SDATA, SCLK, and REQ/LOK from the ICA and one output line FLG/ACK to the ICA. When the ICA wants to

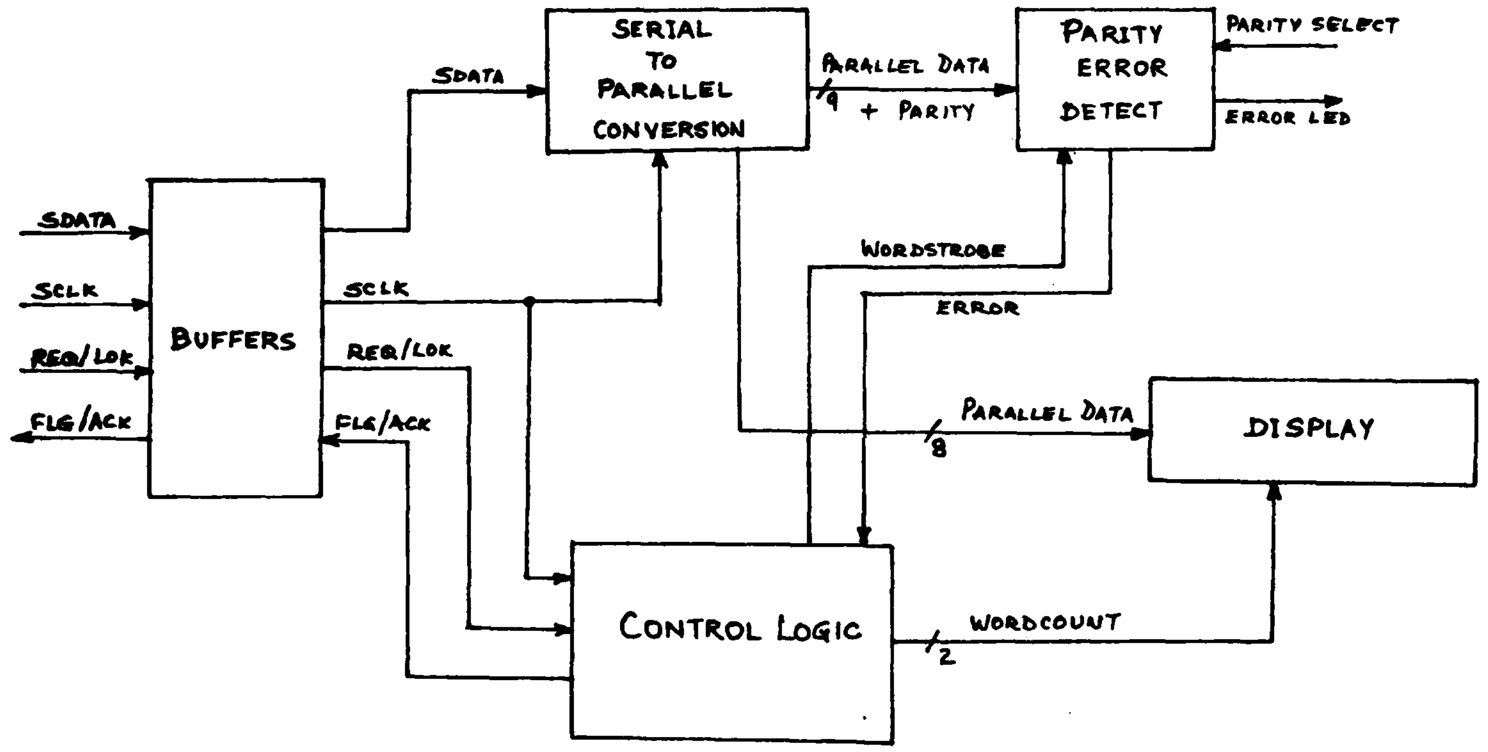


Figure 68 Block Diagram of Serial Receiving Subsystem

display serial data it sets the REQ/LOK line high. The control logic immediately responds by setting the FLG/ACK line high. When the ICA detects that the FLG/ACK line is high it starts sending the clock pulses on SCLK and simultaneously sends data on SDATA.

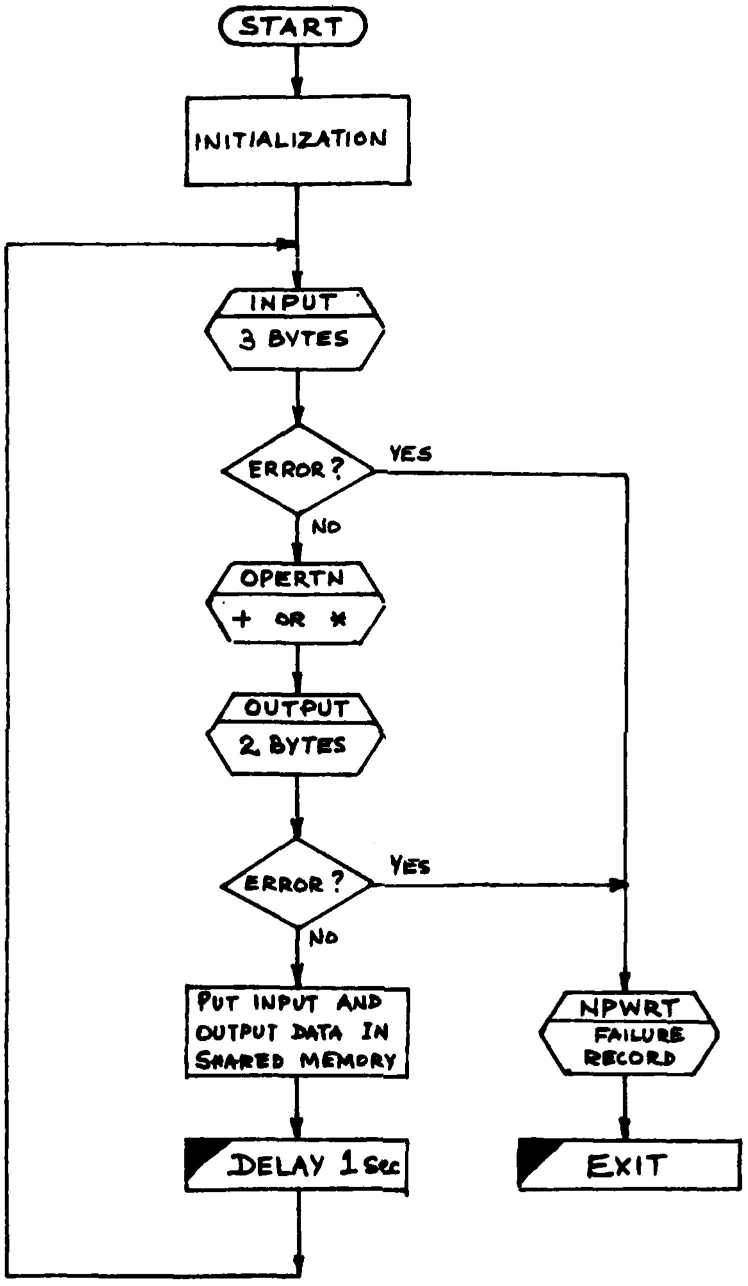
The serial to parallel conversion block accumulates 9 serial bits with the help of the clock (SCLK) and sends them to the parity error detect block. It also supplies the display with the 8 data bits. At the end of 9 bits WORDSTROBE goes high which allows the parity error detect block to determine if there was a parity error in accordance with the parity select switch. Also WORDCOUNT increments and enables the display to latch onto the 8 bits and causes the data to be displayed. If there is a parity error, ERROR will go high and the control logic will set FLG/ACK low, which will stop the transmission. Also the line to the RX parity error LED will go high and light the LED. If no error occurs, at the end of the next 9 bits the cycle will be repeated.

For the subsystem to function properly it must have the sending subsystem connected to one group of the ICA and the receiving subsystem to the other group of the ICA. The connections to the LM are made through the 4 I/O lines at the buffers of each subsystem.

6.1.2 SOFTWARE DESIGN

For the subsystem to function as a calculator there needs to be a software program running in the LM to perform the calculations. This program is stored in the nameplate of the subsystem and runs in the LM as a nonresident task.

A broad picture of the processing performed by this program is given in Figure 69 . A detailed description of the program, subroutines



COMMENTS

Set SM buffer flags

2 bytes for BCD operands, 1 byte for operation

during input

perform operation

result in BCD

during output

Figure 69 Serial I/O Program

used, and common data referenced, appears in Appendix C, Section 6-A. The initialization procedure consists of setting up the shared memory flags for receiving data. Once this is performed the program attempts to input data from the subsystem through the ICA, which must be configured properly prior to running the program. If the input procedure is successful, the program will have three bytes of BCD data. The second byte defines operation: add or multiply, while the first and third bytes define the operands. The program proceeds to implement the operation by calling the math service of the executive. First the BCD operands will be converted to binary and then the operation will be performed. The result of the operation is finally converted to 4 BCD digits. This result is output back to the subsystem through the ICA. If the output procedure is successful, the program transfers the input and the output bytes to the data buffer in shared memory.

If an error occurs during input or output transmission, the program will immediately write a failure record into the subsystem's nameplate. The record will contain the time of failure and the error type. Consequently the program will stop execution.

For this subsystem the ICA needs to be configured as follows: Group A - serial output, Group B - serial input (flag mode). The seven configuration bytes for each group are in the nameplate and it is the duty of the command interpreter to configure the ICA with these bytes, upon receiving the appropriate command.

For a detailed description of the program refer to Appendix C, Section 6-A.

6.1.3 NAMEPLATE DATA

As was mentioned earlier, the nameplate holds the ICA configuration bytes and the software program required by the subsystem to which it is attached. The ICA configuration consists of 7 bytes for each group of the ICA, totaling 14 bytes in all. The software program for the serial subsystem consists of a total of 959 bytes. The header for this program is also present in the nameplate and consists of 13 bytes. Lastly there is a checksum byte at the end of the program. A map depicting how these elements are stored in the nameplate is shown in Figure 70.

If an error occurs during an input or an output request of the subsystem program, a failure record is written into the nameplate. Figure 48 shows the contents of a typical failure record.

6.2 SYNCHRO SUBSYSTEM

The synchro subsystem consists of an enclosure with two synchros. In this subsystem, one of the synchros operates as an input device while the other operates as an output device. The angle of the output synchro is incremented by the angle of the input synchro periodically. This results in a rotational motion of the output dial. The increments are made in 3 second intervals in order to meet the response-time limitations of the synchro. The value of each angle is measured (input) or controlled (output) in terms of three synchro voltages associated with the angle.

6.2.1 HARDWARE DESIGN

The hardware of this subsystem consists of two synchros. It also contains certain connections to the synchro windings which enable the synchros to operate properly.

The synchros used in the implementation have windings with the configuration shown in Figure 71(a). Such a winding configuration charac-

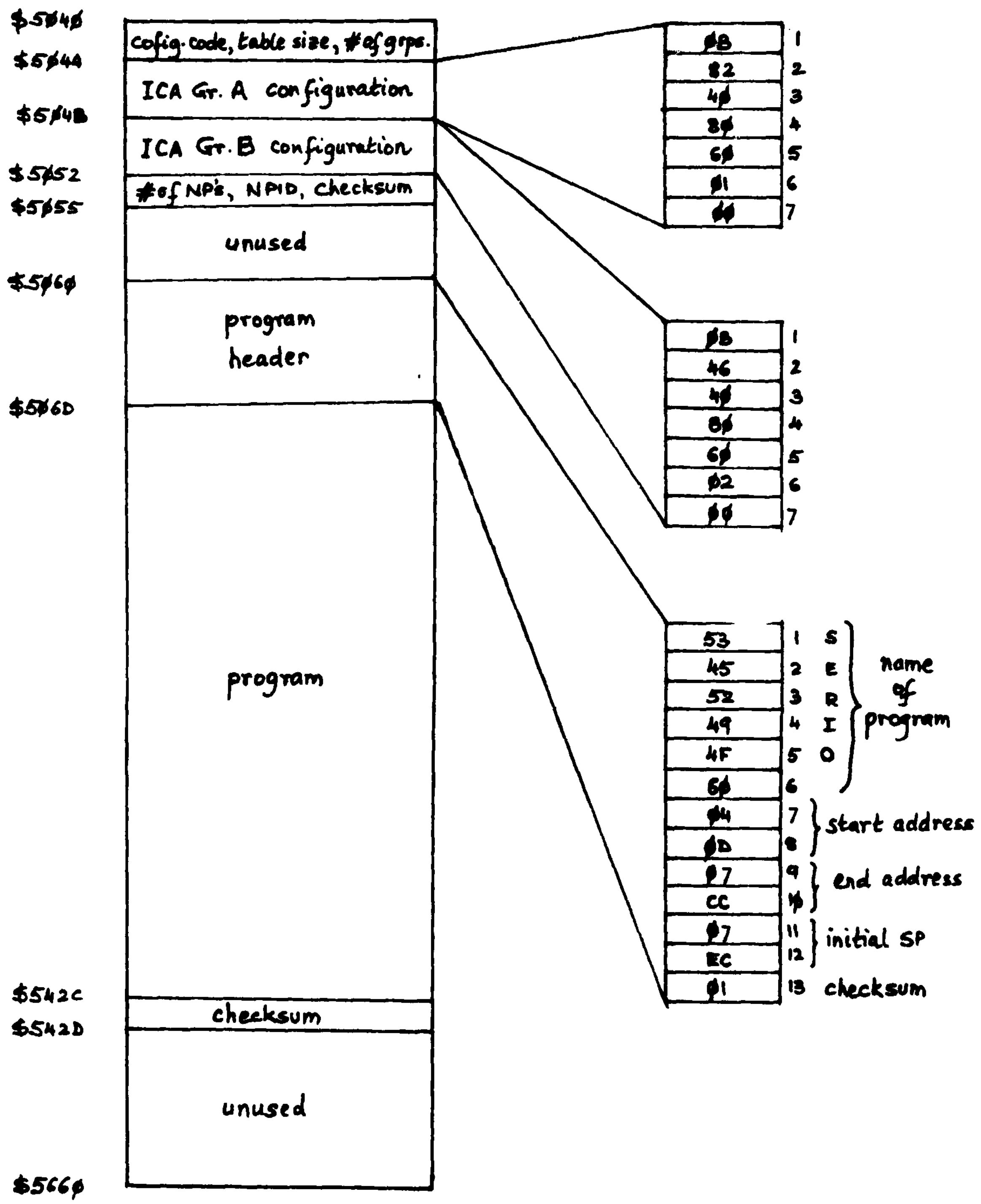
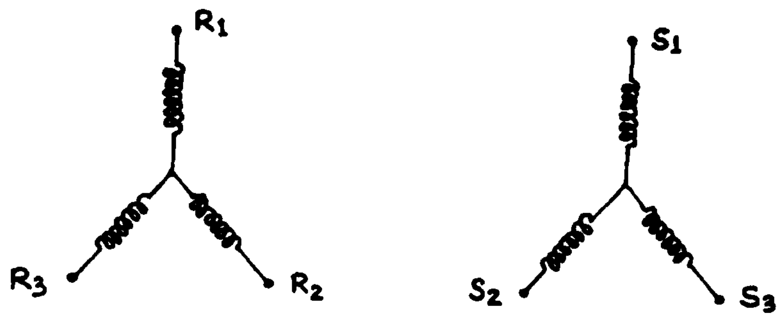
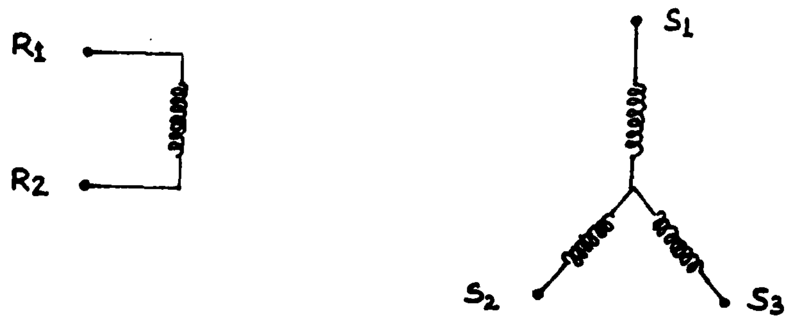


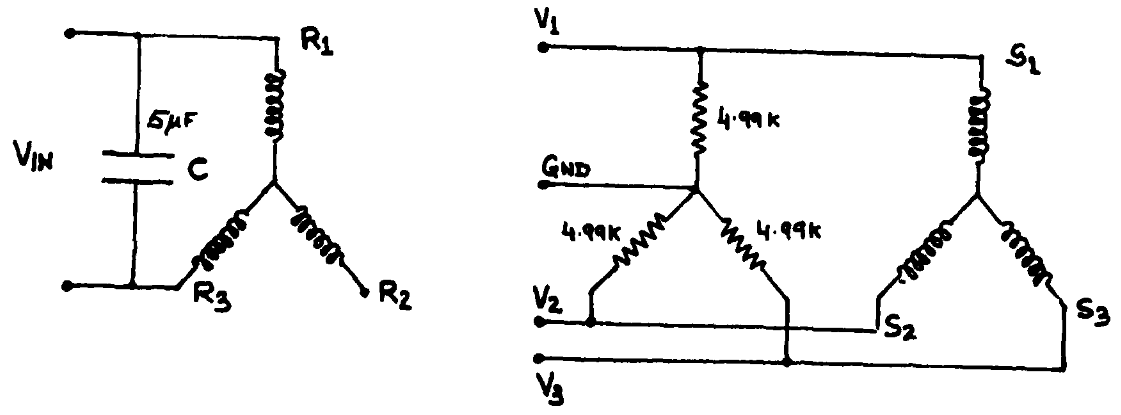
Figure 70   Map of Nameplate Data for Serial Subsystem



(a) Differential synchro



(b) Transmitter synchro



(c) Modifications implemented

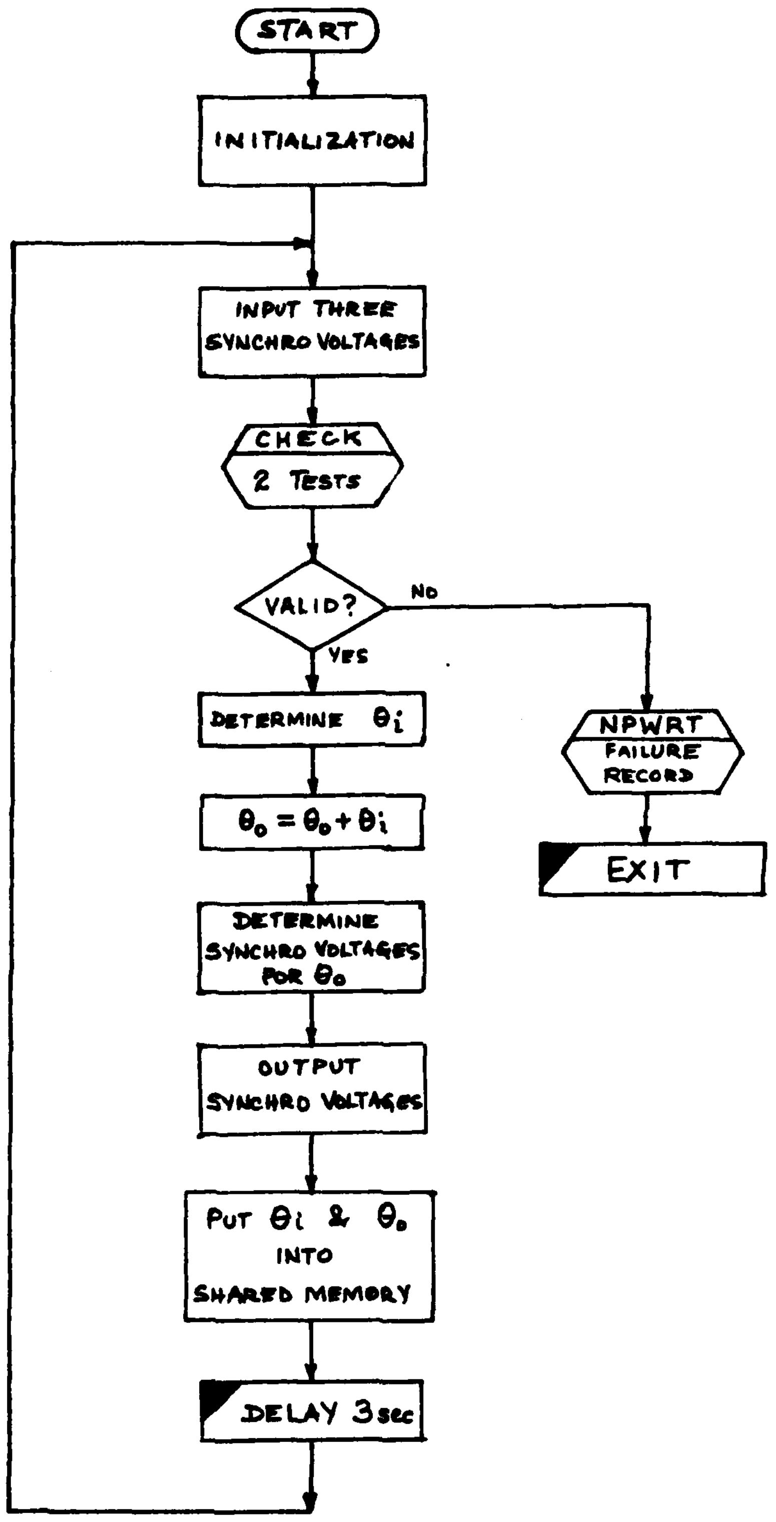
Figure 71 Synchro implementations

terizes a differential synchro, which is not suitable for this application. What is desired is a transmitter synchro with windings in the form shown in Figure 71 (b). Also, the voltages supplied to or sensed from the stator windings should correspond to phase voltages. However, the synchro stator does not have an accessible ground connection. Therefore a ground is created through a resistor network between the windings. These modifications are shown in Figure 71(c) which shows a winding wiring which is equivalent to the synchro shown in Figure 71(b). The capacitor between the two rotor windings is used to reduce the power factor of the windings.

6.2.2 SOFTWARE DESIGN

As mentioned earlier, to demonstrate the functioning of the subsystem, the angle of the output synchro is incremented periodically every 3 seconds. The value of each increment is determined by the angle set on the input synchro. To perform this demonstration a software program must run in the LM. This program is stored in the electronic nameplate and is loaded into the LM as a nonresident task.

A broad picture of the processing performed by this program is given in Figure 72. A detailed description of the program, subroutines used, and common data referenced, appears in Appendix C, Section 6-B. The initialization procedure consists of setting up the shared memory flags for receiving data and initializing the ICA for synchro operation. Once this is done the program inputs the three synchro voltages of the input synchro through the ICA. The voltages are input in the form of three single byte binary values of an ADC. The three voltages are tested to determine if the synchro is malfunctioning or not. The first test checks if the sum of the three voltages is zero. The second test calculates the value of



START
INITIALIZATION
INPUT THREE SYNCHRO VOLTAGES
CHECK 2 TESTS
VALID?
NO
YES
DETERMINE θ_i
$\theta_o = \theta_o + \theta_i$
DETERMINE SYNCHRO VOLTAGES FOR θ_o
OUTPUT SYNCHRO VOLTAGES
PUT θ_i & θ_o INTO SHARED MEMORY
DELAY 3sec
NPWRT FAILURE RECORD
EXIT

COMMENTS

Set SM buffer flags

Check if voltages valid

θ_i is input synchro angle

θ_o is output synchro angle

Figure 72 Synchro I/O Program

the synchro constant 'A' by calling upon the executives math service, and compares it with the nominal value. If the voltages fail any of these tests, the program writes a failure record into the subsystems nameplate and exits. A typical failure record is shown in Figure 48 .

If the tests are successful, the program determines the input synchro angle from the three voltages by using the math service of the executive. It then adds this angle to the previous angle of the output synchro. Next it uses the executive's math service again to determine the three synchro voltages corresponding to the resulting output angle. Finally the program outputs the three synchro voltages to the output synchro, writes the input and output angles into SM, and waits for 3 seconds before it repeats the procedure.

6.2.3 NAMEPLATE DATA

The nameplate holds the ICA configuration bytes and the software program required by the subsystem. The configuration is 7 bytes for each group - 14 bytes in all. The software program for the synchro subsystem consists of 1061 bytes. The header for the program is also in the nameplate and consists of 13 bytes. A map depicting how these elements are stored in the nameplate is shown in Figure 73 .

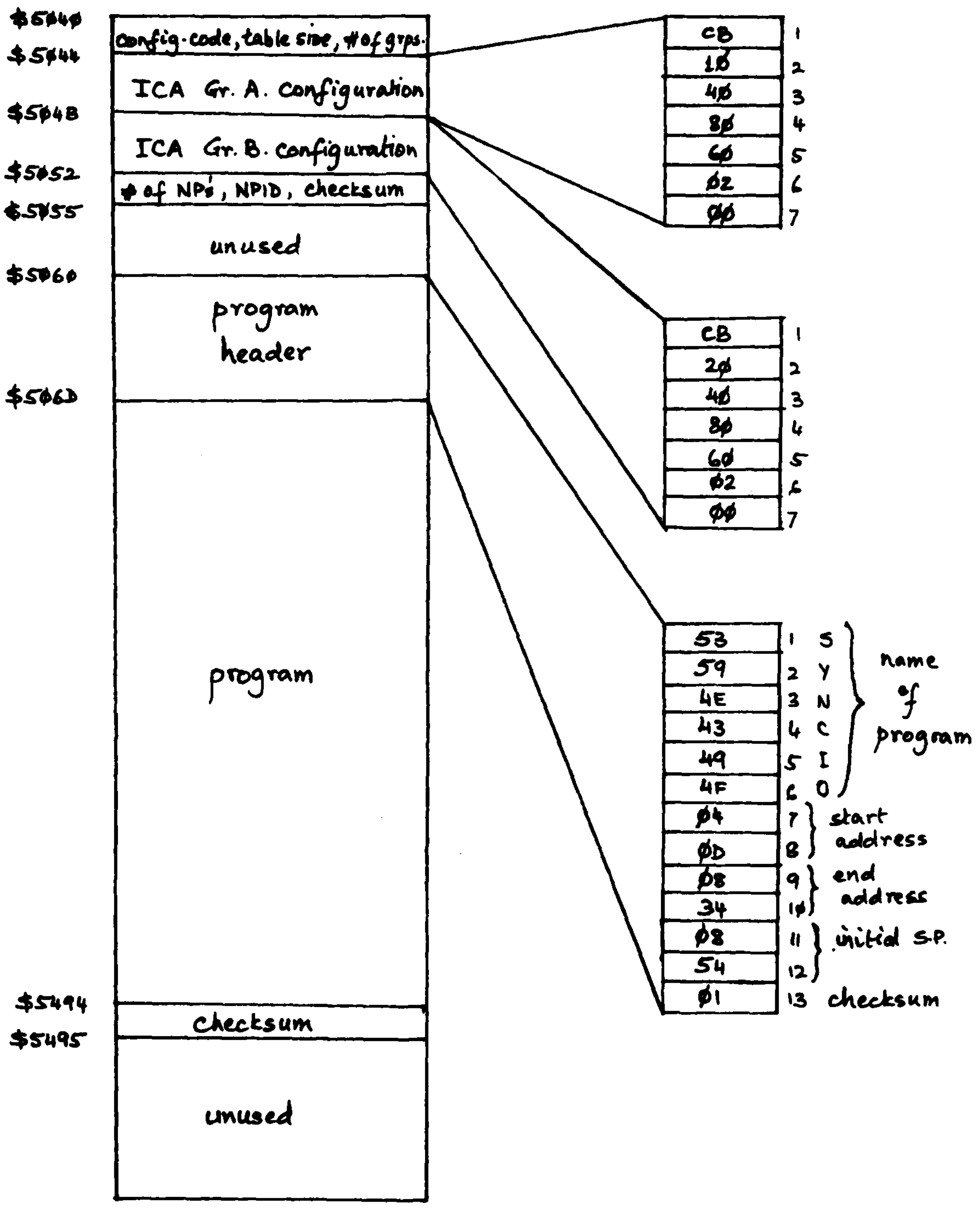


Figure 73 Map of Nameplate Data for Synchro Subsystem

modes - flag or refresh. In the flag mode data is transmitted only when the STBSW switch on the panel is closed. Prior to closure of STBSW the value of the thumbwheel switches can be set. On closure of STBSW, data on the thumbwheel switches is latched until either the transmission is successful, or the RESET switch on the panel is closed. In the refresh mode data is transmitted as soon as it is requested by the LM. Data is latched only as long as the transmission takes places. Changes on the thumbwheel switches are detected during the periodic transmission cycles initiated by the LM.

The block diagram of the sending subsystem is shown in Figure 67. A detailed drawing of this circuit is presented in Appendix B, Section 6-A. block diagram there are two inputs REQ/LOK and SCLK from the ICA and two outputs SDATA and FLG/ACK to the ICA. The ICA requests transmission by